Project Number: MAD-006A

CENTRALIZED INTRUSION DETECTION via SWARM ROBOTS

A Major Qualifying Project Report

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by:

Nathan Fuller

Nathan Rosenblad

Christopher Thein

Christopher Warms

Derek Williams

Date: March 1, 2007

Approved:

Professor Michael Demetriou, Major Advisor

Acknowledgments

We would like to especially thank Professor Michael Demetriou for all his guidance throughout this project. Special thanks are extended to Eric Twark, whose extensive knowledge of programming, aided us in the creation of a program for the robots. We would like express our sincere gratitude to Nathan Rosenblad whose efforts in creating basic circuit board design, procuring parts, and base construction over the summer made this project possible. Other special thanks are sent to the Aerospace Engineering Professors who shared their input and ideas at our weekly meetings.

Abstract

The goal of this project is to design, construct, and implement a centralized system to control drone robots equipped with visible and infrared light sensors which systematically detect, track, and contain an "intruder." The drone robots and base station use custom-written software which allows wireless intercommunication and control between them via radios. Once the program commences, the robots are controlled autonomously by the base station; there is no human input, other than controlling the "intruder's" trajectory.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

The goal of this project was to develop a network of robots with the capability of detecting an intruder, and having the ability to track, follow, and effectively surround it. Prior to the development of this system, other robotic programs used by the military and other universities were studied and reviewed. Previous autonomous robot models were examined, as this project aims to add more advanced capabilities to autonomous technologies already in place. An autonomous robot network was developed to allow a robot to detect and capture an intruder alone and/or with the aid of other robots.

This was one of the first autonomous robotic projects to be completed in WPI's Aerospace Engineering Department. WPI was not the first to investigate the need for autonomous robots, as there have been many university and military research projects investigating the plausibility of utilizing self-guided systems. One such system is the military's "Predator," which is an autonomous unmanned aerial vehicle, requiring human input only on certain mission critical decisions.. It requires the input of a directive or target that it aims to achieve or destroy.

Figure 1: Unmanned Aerial Vehicle, "Predator"

The "Predator" has the capability to make its own decisions in flight as well as to communicate to other "Predators." The concept of a completely autonomous robot is complex, but beginning with basic sense-and-react algorithms, as was conducted in this MQP project, one is better able to understand "the next step" in achieving a fully autonomous system.

Another project that involved autonomous robots was the “Multiple Autonomous Robots” (MARs) program conducted at *GRASPP Laboratories,* PA. The MARs project worked with multiple robots over various types of terrain using several types of sensors as seen in Figure 2. These sensors included infrared distance sensors, omni-directional cameras, video transmitters, and powerful onboard processors. This project modeled tests that involved random terrain and hazardous conditions (smokey buildings, inclement weather). They used laptop processors to increase the robots’ operating capacity as well as sensor functionality. The methods implemented by the MARs project provided useful information on some of the characteristics that needed to be investigated for the model being developed in the WPI project.

Figure 2: MARs Robots Navigating Obstacles

The research completed by the MARs group showed that there was the need for an autonomous system of robots with the ability to track, capture, or contain an object that makes a safe zone unsafe. This object can be a physical mass or a liquid that results in the harmful contamination of previously clean area. There are a variety of sensors that can be used to track an intruder, (video, light, distance, vibration, chemical, or biological, etc). The robot network was developed to detect an intruding contaminating source using only distace and light sensors. The project was limited to this scale of sensors due to the availability of equipment and budget. The sensors used in this project provided an excellent proof of concept in creating an autonomous network resulting in the potential to integrate additional sensors in the future. The light sensors that were integrated into the detection robots provide for the tracking of the intruder which emits a contaminant signature of blue LED light. Using the robots that were built, along with Cubloc® software, the sensory input from the robots is transmitted wirelessly to the main base station and in turn to any other robots connected wirelessly with the same software. The program’s ability to

communicate wirelessly is the key to allowing multiple robots to be used in the tracking and capturing of said intruder.

Movement is a key part of the robots' operational features. Each robot is required to tabulate its own position in order to effectively communicate it to the others operating in the area. Various transportation methods such as flying, wheeled motion, tread movement, or roller balls, each have their own benefits but also require different methods for calculating position. The choice of movement was decided in conjunction with the means of determining position. Wheel based mobility was chosen due to the fact that a proven system was already in place that was accurate and useful for the scope of the project.

Position calculation was another key component of the project, as it was an integral part in tracking not only the location of the intruder, but the positions of other robots as well. Accurate movements needed to be recorded in order to effectively track a robot's position. The tracking information was relayed to the other robots through the base station allowing for combined efforts between the robots to contain the intruder. The environment in which the system was designed to operate was on a small scale, therefore positioning technologies such as GPS did not provide the accuracy and also had prohibitive costs. The localization method used required the installation of an optical rotary encoder on the drones' drive motors to track the number of wheel revolutions completed. This method was been proven to be accurate using the Cubloc® software, and was implemented in the design of this project.

At the time that this project was conducted, there were no other existing projects at WPI that utilize 9 robots that communicate together in an attempt to trap or contain an intruder. This project breaks a lot of new ground in the field of autonomous inter-communication, position tracking, and robot controls. The scope of this project encompassed a large number of aspects that were challenging to complete in the 3 term period allotted for the project.

The developed program effectively executes the utilization of the robots' sensors as well as allows the robots to communicate with one another, resulting in an efficient, expeditious containment of an intruder. The design of the system allows the capture of an intruder in multiple scenarios in various differently sized areas. The project presents a basic model of autonomous robot based communication and its effectiveness in completing the specifically tasked functions. It also presents great opportunities for future students to expand and further develop the tracking and containing methods as well as programming of robot intercommunication.

2. COMPONENTS

The following sections describe the components of the robots that were purchased or manufactured and assembled. Additionally, this section details what initial components were necessary to be built prior to initial testing of the robots. Certain parameters were assumed, as they provided guidance on how the project was developed in certain areas. For example, the distance sensors maximum range is five feet, so the testing environment was limited to no more than 50 square feet. Therefore, the robots would eventually contain the intruder, but after an extensive amount of time. Limits such as these provided more defined requirements for the development of the system.

2.1 Testing Environment

Defining acceptable testing spaces helped concentrate the project goals. The fifty square foot dimensions of a rectangular area were assumed for a number of factors:

a. Battery life of the robot
b. Capability of sensory equipment (range)
c. Timing of tracking intruder
d. Accessibility to malfunctioning robots
e. Number of robots used for scenario

Two main reasons for the definition of a testing space included the fact that the sensory equipment used had limited capabilities, and the amount of time required to track and surround an intruder needed to be within limits of detecting robot's battery life. In a large-scale application of this model, a robot may be given hours to detect an intruder over a large distance, and provided with the proper equipment to scan large areas. With the allocated budget and time restraints for developing the model system, the smaller operating area was necessary and well suited for the robots' abilities. The smaller area also provides for the recording of data that can be used to verify the accuracy and effectiveness of the algorithmic functions and the system's responses.

2.1.1 Environmental Test Platform (ETP)

Due to the fact that the speed (eight inches per second) and physical size of the robots did not mandate a large operating area to demonstrate initial results, an original testing space of 6 by 9 feet was defined. A platform was designed and built, keeping in mind the necessity to access the robots during testing (all sections of the platform were required to be within reaching distance). By first operating the system in this well-defined controlled are, it allowed for the decision to be made as to whether or not the model was ready for testing in a larger area. The flat, smooth surface eliminated unknown variables such as reflection and slippage. It also allowed the addition of temporary obstacles which were used to test the maneuverability functions of the robots. Once moved to a larger testing area, variables such as smooth, flat surfaces, and external interference (lights, wax floors, reflections) were no longer controlled. For initial testing, the surface presented no obstacles or obstructions that affected the robots' distance and light sensors. With no foreign impediments on either of the sensors (light or distance), the ability to accurately track the results of any test was achieved.. To minimize the number of impediments, the platform was constructed with a plain white base. The base consisted of a 3/16 inch rigid particle board material with a semi glossy white surface finish as seen in Figure 3 below.

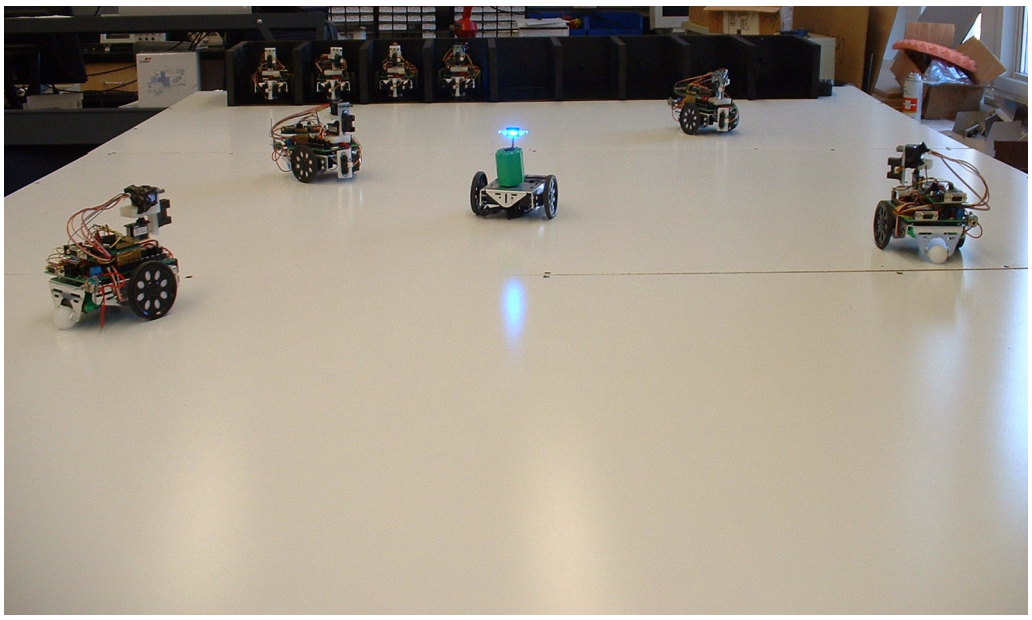


Figure 3: Environmental Test Platform with Robots and Intruder

The platform was required to be mobile, giving the robots the ability to be presented or demonstrated in various locations. This was taken into consideration, as the platform was designed in three identical sections. Each section had the dimensions of 3 feet by 6 feet by 3 inches (length x width x height). The sections were bolted together to provide stability and eliminate cracks between the sections of the whiteboard.

2.2 Drone Robots

There were ten drone robots that were assembled for this project. Nine of the robots were designed for tracking and sensing an intruder, and one of the robots was set as the intruder. The setup of the main circuit boards of the robots was designed during the summer previous to the official project by Nathan Rosenblad, who was participating in a summer internship. Of the ten robots, the intruder is the only one not equipped with the long/short distance sensors and light sensors. As seen in Figure 4, the remaining nine robots have the following components:

a. Radio communication board
b. Standard battery pack
c. Power saving feature
d. Infrared distance sensors
e. Boe Bot chassis
f. Wheel based movement

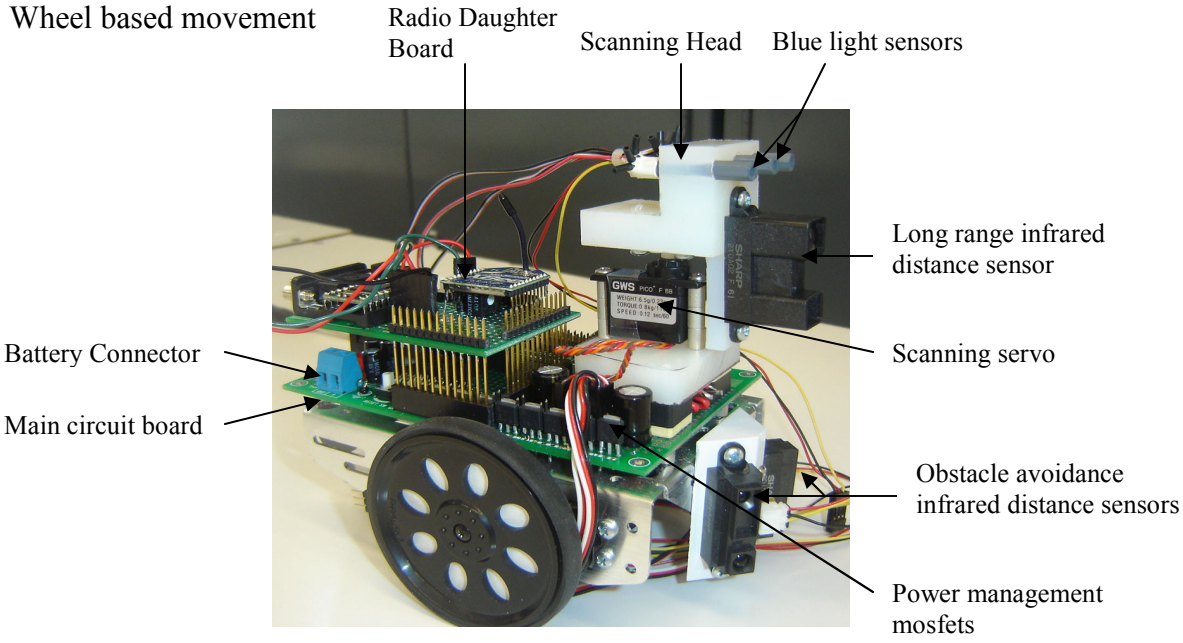


Figure 4: Fully Assembled Robot

Each component above was used during the testing of the robot. Some of the capabilities were modified from its manufacturer's specifications to be more efficient for this project. The nine drone robots are equipped with a set of long and short range infrared sensors that detect distance to objects and light. The purpose of these devices was to detect other robots, obstacles, walls and sources of contamination.

These sensors are divided into two separate groups: navigation sensors and intruder detection sensors. The navigation sensors are proprioceptive in that they are responsible for the functions that are inherent to the robots such as the compass or positioning sensors. The intruder detection sensors are exteroceptive, in that they are influenced by outside sources such as the light from the intruder or distance readings induced by a physical obstacle.

2.2.1 Radio

Each robot was equipped with a 2.4 GHz radio with a range of 490 feet, allowing communication from a base station to the robots. The base station consists of a PC operating the Cubloc® software and the program that was developed on it. The antenna allows wireless communication between the robot and the station. The program was designed to directly communicate from the station to the robots or vice versa. Information from the computer program is sent to the robot to order it to engage certain functions. The robots also operate based on the program embedded in their primary CPU, and relay gathered data to the base station. The robots have the potential to communicate robot to robot, but due to limitations in the onboard processors, this type of delegation is not viable. The radio daughter board and its components can be seen in Figure 5.

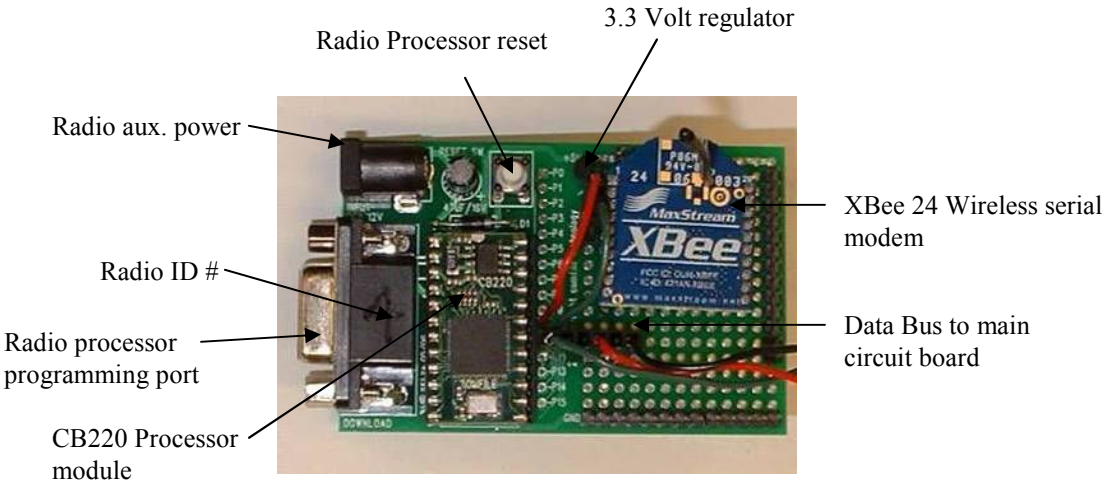


Figure 5: Radio Circuit

2.2.2 Battery Pack

The onboard power supply for the robots is a Nimh (Nickel Metal Hydride) pack containing 6 AA sized batteries. Each robot is equipped with a rechargeable battery pack mounted to the underside of the robot chassis. To recharge the batteries, the battery packs can be wired to a battery recharging unit, or can be directly connected to a power supply providing a charge of 9 volts. With the addition of an "on/off/charge" switch, as seen in Figure 6, one can stop charging the batteries easily when it's wired directly to a power source.

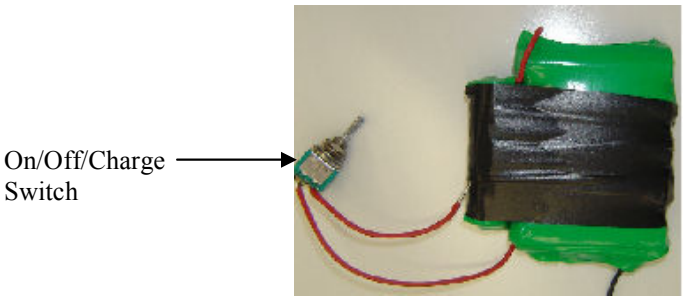


Figure 6: Battery Pack with (On/Off/Charge Switch)

The switch has three positions: up (on), middle (off), and down (charge). It takes the battery pack about 10 hours to charge while in 'charge mode'. The battery charger was attached to the chute which provided protection to the robots while they were being charged. These batteries are suitable for this project because if they ware run at max capacity, meaning all functions of the robot were running constantly, the battery life during this operation would be about 4 hours. There was no need to run all the functions of the robot at once so the battery life increases to about 8-10 hours.

2.2.3 Power Saving Feature

Similar to what was stated in the 'battery pack' section above, the robots have the capability to use only the functions necessary to complete the required operations. The program developed to control the robots has commands built in that command the robot to perform specific functions when certain variables are encountered. This feature allows the robot to sustain a longer battery life because it knows which functions are necessary at any given time.

2.2.4 Infrared Distance Sensors

Each drone robot is equipped with a pair of identical SHARP GP2D120 infrared object detectors. These detectors, hard-mounted on the front of the robot have the ability to sense objects in the range of 5 to 40 cm from their front faces. The use of two detectors mounted side by side allowed the onboard processor to make movement decisions in a given direction based on the feedback from these detectors.

A long range infrared sensor was also part of the contamination detection system. The GP2Y0A02YK, by Sharp, is capable of detecting objects in the range of 20 to 150 cm from the front of the sensor. One of these long range sensors was mounted on each robot. They served to aid in contamination detection by locating the distance to the source of the blue light contamination. The long range distance sensor is shown in Figure 7.

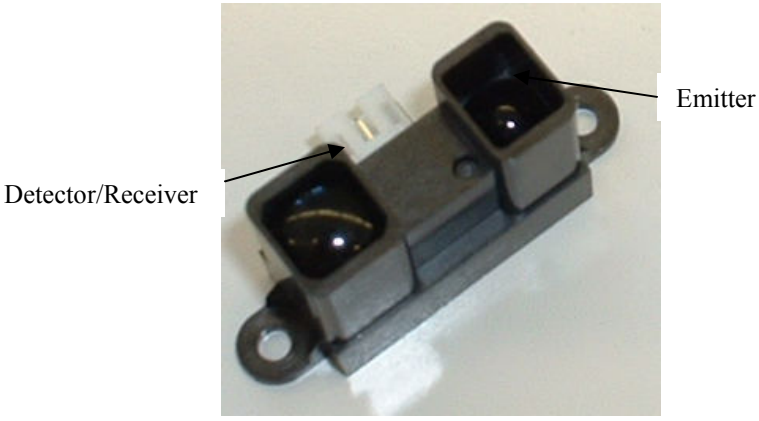


Figure 7:GP2Y0A02YK Infrared Distance Sensor

2.2.5 Boe Bot Chassis

The base platform chosen for the drone robots was the Boe Bot® Chassis by the Parallax Company. The chassis was a standard part that allowed for easy integration with the electronic subsystems. Constructed of aluminum, it had pre-cut square openings for attaching servo motors for mobility, in addition to a number of standard holes for mounting electronics, wheels, and sensors. Three additional holes were made to this chassis to support the main circuit board. The chassis is shown in Figure 8.

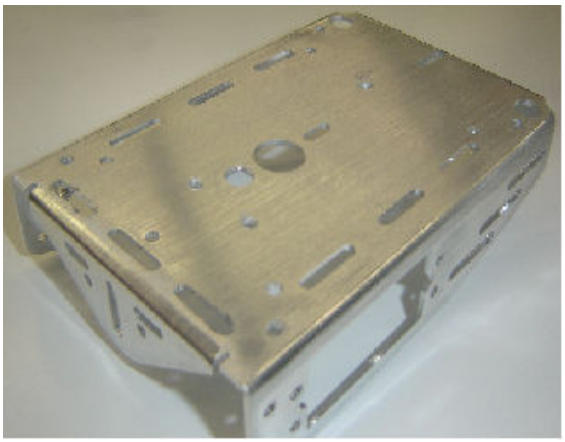


Figure 8: Boe Bot® Chassis

2.2.6 Wheels

There was a choice of using wheels or treads as the method of movement for the robot. Although treads provided a lower amount of slippage compared to wheels, the final decision was made to use wheels. Initially, treads were installed on the robots; however, the method of positioning that was attempted was not successful due to the low resolution of the tread counters that were mounted to the drive sprockets. Wheels were finally chosen because there was already a position tracking algorithm in place that was modified to work better for our model. With the wheels, operation in rough terrain such as gravel or shag carpet is not feasible, but the amount of error in the wheels is far less than what could be obtained with treads. An example of the wheels that were used is shown in Figure 9.

Figure 9: Wheel with Encoder Cover

2.3 Chute

The chute was designed to give the robots a starting point for initialization of all detection scenarios. Since the robots' positioning system is absolute, they need to have a known starting point, or global origin, allowing the base station to relay accurate position of where an individual robot is positioned in the operating environment or in relation to additional activated robots. The chute provided this known starting point due to the fact that it is placed on the field in a known position, and the position of each robot in each chute was known. The position of each robot is based off the bottom left corner of the chute. Since the dimension of the chute was known, the area of the test platform was known, and the placement of the robots were known, the robots global origin is defined and its location anywhere on the platform is known with respect to the origin. This initial set up was the only user interface required to get accurate position. Once the original position was calculated and entered into the simulation program, the robots were able to track themselves and the other robots for the duration of the test.

The secondary function of the chute was to operate as a battery charging station. Holes were drilled into the rear wall of the chute to allow a charging connector wire access to each of the robots. The robots, once finished with their testing, returned to the chute and were manually plugged in by the user. The chute is shown in Figure 10 below.

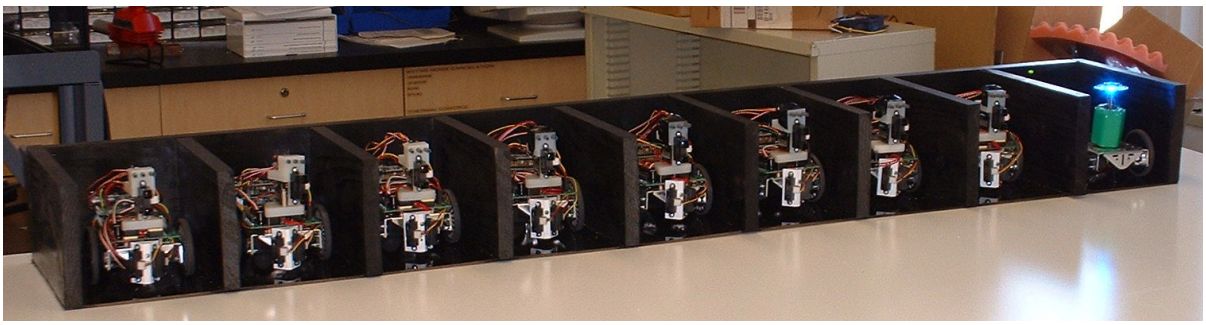


Figure 10: Chute: The origin and charging port of the robot

3. SUBSYSTEM DESIGN

The "Subsystem Design" section explains how components previously described in the components section above were integrated resulting in a high performance intrusion detection network. The designs below were specifically manufactured for this application. These subsystems include the panning sensor head, LED contaminant robot, optical positioning system, performance coding, and the main circuit board design.

3.1 Panning Sensor Head

In order to locate and track the contaminating intruder, a sensor head was designed on which both the long-range infrared object detector and three light-to-voltage converters are mounted. The infrared distance sensor was mounted vertically as seen in Figure 11 to allow the scanning capability that is an integral part of locating the intruder. The sensor scans in a 180 degree vertical plane. Many design iterations were performed and the final design was chosen due to the fact that it successfully integrates its components for use in the dynamic testing environment. If the future testing environments change, it may be necessary to modify the sensor head to acquire the best results for the new environment.

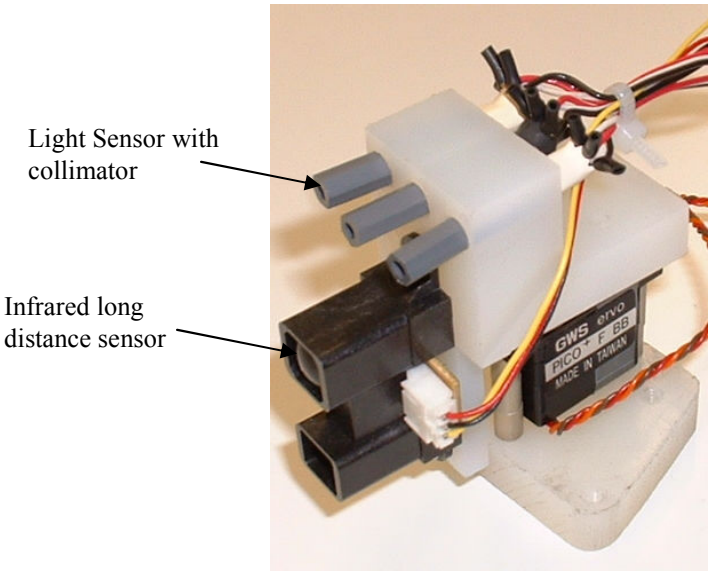


Figure 11: Panning Sensor Head with Sensors

3.1.1 Light Contamination Detectors

The drone robots are each equipped with a set of sensors that serve the function of detecting a remote contamination source. For the applications that were investigated in the project's current research, the contamination source was a blue light emitting diode (LED) attached to an independent robot. The LED, mounted to the "intruder" robot, emitted a blue light (approximately 470nm wavelength). Therefore, each drone robot was equipped with a set of sensors capable of detecting this wavelength. The sensors used to detect the contaminating source were classified as high sensitivity light to voltage converters. The TSL 257 had its highest response in the 350 to 500 nm wavelength range, with its peak at 590 nm (TAOS). With an input voltage source of 2.7 - 5.5 volts, the sensor outputted a voltage that was directly proportional to the light intensity.

Initial tests showed that the sensors were reading values that were close to the maximum limit when used unmodified in a room with ambient fluorescent lighting. This was due to the fact that the sensors, in their original form allowed light to enter from 180 degrees around the front surface. Through the attachment of a collimating tube to the front surface of the sensor, the amount of ambient light that entered the sensor was limited. It only permitted light rays that were traveling in a straight line parallel with the tube to enter the sensor. This allowed the drone to determine the location of the intruder based upon the angle with the maximum blue light levels. The TSL257 light sensor and collimator tube can be seen in Figure 12.

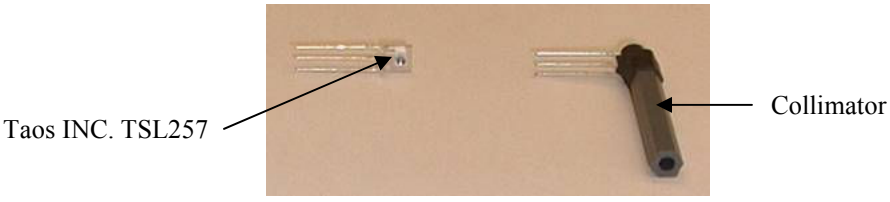


Figure 12:TSL257 Sensor with and without Collimator

3.1.2 Angular Positioning of Light Sensors

The angular placement of the light to voltage converters in the panning sensor head was determined through the use of initial experimental data. A set of calibrations was completed on a single light to voltage converter (with collimator) to determine its field of view.

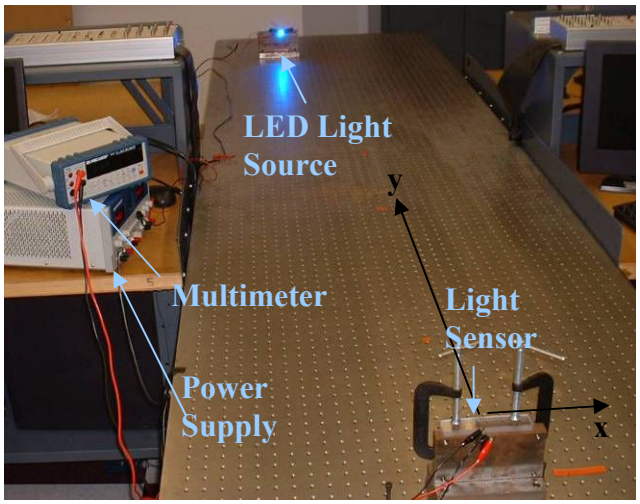


Figure 13: Light and Sensor Calibration Setup

The algorithm developed to locate and track the contaminating light source requires known calibration values for the field of view of each individual sensor, as well as the sensor head as a whole. An initial experiment was performed to determine the field of view of a single sensor with collimator attached. As seen in Figure 13, above, a high power blue LED was used as the light source. The experiment was performed on a vibration isolation table that has bolt holes every inch on its top surface. A dual output power supply at 4.0 Volts DC was used to supply the voltage for the LED and sensor. A digital multi-meter was used to measure the sensor output. The LED was mounted to a steel support block and moved to different x and y coordinates of the table, with the sensor output being recorded for each location. This data was, in turn, used to calculate the angular field of view of an individual sensor. A plot of one set of data recorded can be seen in Figure 14, below.

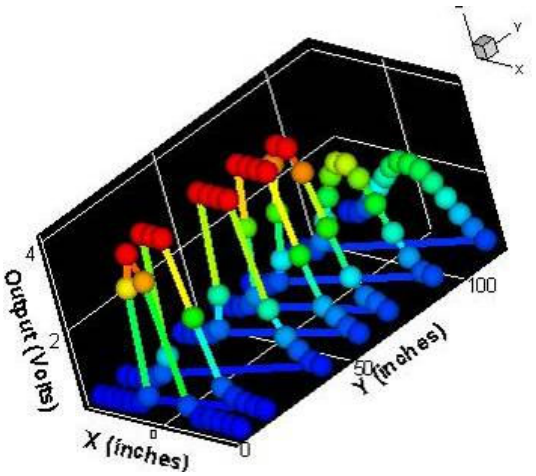


Figure 14: Sensor Output Values

Using the data obtained in the measurements described above, the field of view for the sensor was calculated. During the time of experimentation, the sensor output with ambient lighting (no LED) was 0.479 volts. In order to determine the angle at which the sensors received a noticeable light change, a minimum threshold value for measured light had to be set. This threshold was rounded up from the ambient level to 0.5 volts in order to eliminate any small offsets or outliers. The lowest values above the 0.5 volt threshold were then used to plot a line and determine its angular offset from the centerline of the sensor. A plot of the minimum over-threshold values with overlaid linear trend lines is shown in Figure 15 below.

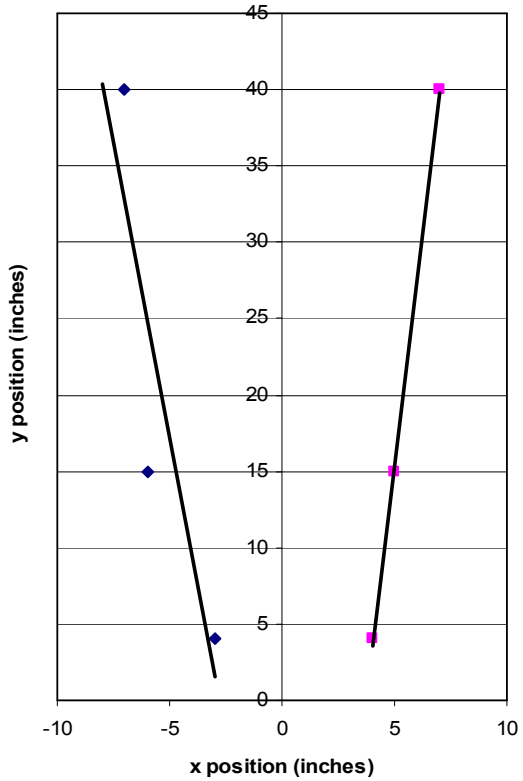


Figure 15: Plot of Minimum Over-Threshold Data

It can be seen that there is a bias in the data values towards the positive x-coordinates. This is most likely due to an angular offset in the mounting of the sensor on the steel support block. This offset is ignored due to the fact that the field of view is taken as the combination of the two angles. The field of view angle for the sensor-collimator unit was estimated using the slopes of the linear trend lines shown in the plots above ($m_L = 7.77$, $m_R = 12.071$).

$$\Theta = \left[\tan^{-1}\left(\frac{1}{m_L}\right) + \tan^{-1}\left(\frac{1}{m_R}\right) \right] = 12.07\,\text{deg}$$

Using the field of view angle (Θ), the offset angles for the side sensors were determined. By setting the intersection point for the side field of view with the center field of view to be four inches, the angular offset angle (θ) is calculated using the following equation:

$$\theta = 90 - \tan^{-1}\left(\frac{d}{d\tan\left[\frac{\Theta}{2}\right]}\right) - \frac{\Theta}{2},$$

where *d* is the distance to the intersection point from the front of the collimator. The angle θ that was calculated for the distance (*d)* of four inches is 6.93 degrees. Due to manufacturing constraints, the angle was rounded down to 5 degrees. Using a Bridgeport Milling Machine in the shops of Higgins Labs, angle blocks were used to drill holes in the sensor head at this angle and were only available in five degree increments. This angular orientation was tested and proved to be successful. Figure below illustrates the variables used and the arrangement of the light sensors.

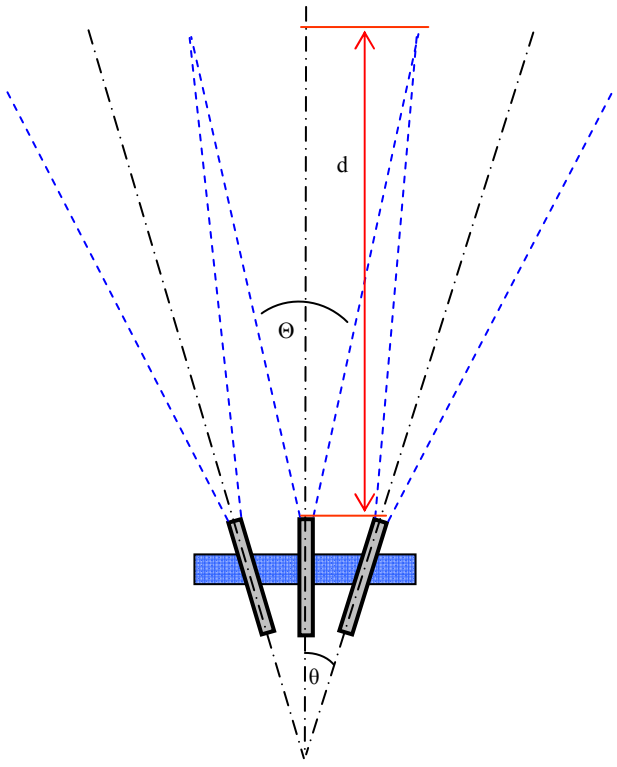


Figure 16: Diagram of Sensor Head Angular Offset Nomenclature

3.1.3 Final Design

There were many alterations to the original sensor head design to accommodate certain modifications to the robot. It was determined that the two axes of sensor head panning ability were unnecessary and possibly detrimental to accuracy, therefore the head was modified to pan along one axis. The two axes of servos increased the uncertainty of observations because the long range sensor cannot take accurate measurements at a downward angle. In addition, project costs were reduced and it freed up more CPU and battery power.

There were other alterations that improved the overall performance of the sensor head. The original design had three separate components required to mount the servo onto the sensor head. It was modified to a single piece sensor head to accommodate the single axis operation, and strengthen the part. This allocated less moving parts, provided more sturdiness and resulted in less induced error.

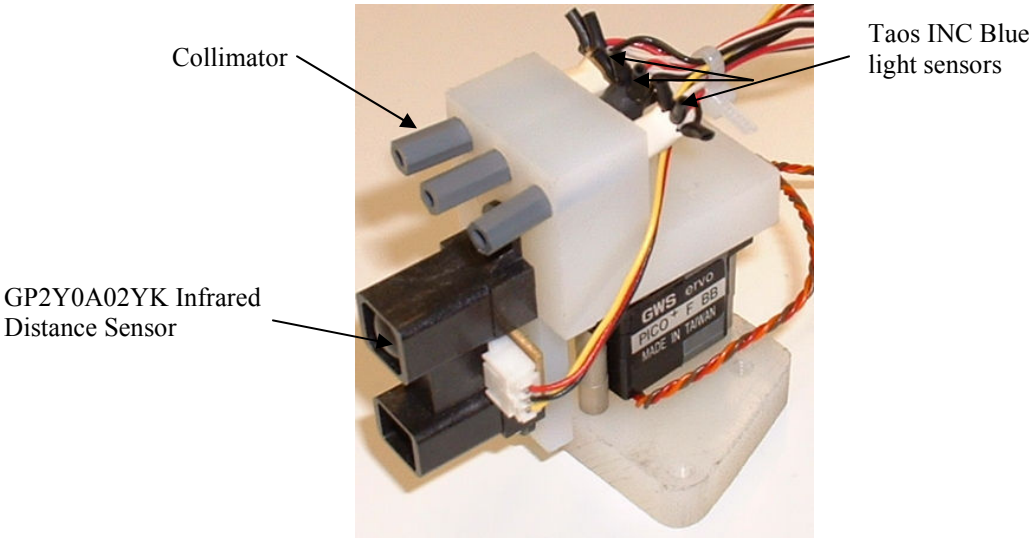


Figure 17: Final Design Panning Sensor Head

One of the major design changes to the sensor head involved the long-range sensor orientation. The sensor was repositioned from a horizontal to a vertical orientation, as is seen in Figure 17 above. This was selected because when mounted horizontally, the IR signal interfered with the short range sensors. The vertical orientation negated the interference, and through

experimental observations this orientation provided measurements that were far superior to previous ones. This phenomenon occurred because the vertical sensor was now directly aligned with the contaminant source. This modification consequently allowed distance measurements to the source of the light to be accurate and properly aligned.

3.2 Short Distance Infrared Sensors

These sensors also needed to be mounted so their scanning area is a 180 degree vertical scan. The two sensors mounted on the front of the robot needed to be mounted at a 45 degree angle outwards (away from each other). This was done in order to eliminate readings from the sensors concerning the same detected object which would skew the results transmitted to the hub and send the robot in the wrong direction.

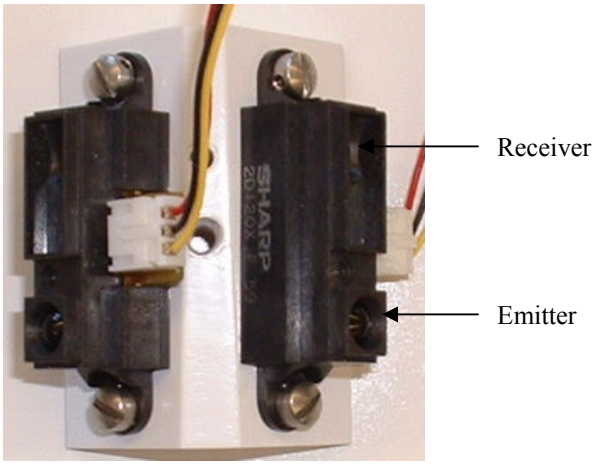


Figure 18: GP2D12 Infrared Short Range Distance Sensors

3.3 LED Contaminant Source

The role of the intruding contaminating source was fulfilled by a simple robot similar to the drone robots. It uses the same chassis as the drones, but was not equipped with a Cubloc® processor. The robot followed a designated path, and was not designed to operate autonomously as the drones. The intruder distributed a contaminant within the 2-dimensional operating environment. Blue light emitted from an LEDs attached to the intruder robot was used as the contamination source and was detected by the sensor arrays of the drone robots. The LEDs emitted light with a wavelength of $\lambda \sim 470$ nm. A key requirement for the intruder was that the robot was able to distribute LEDs light 360 degrees around the robot. This allowed the drone

robots to identify and accurately determine the location of the intruder. Several methods of distributing the LEDs light around the robot were proposed, including: mounting several LEDs around the robot, and using one high power LEDs with a reflective distributor. The idea of mounting several LEDs around the perimeter of the intruder induces a greater error in determination of the target's location due to the fact that there are multiple light sources on the same intruder that were detected by the drone robots. The development of a light distribution device allowed the light from a single high power LED source (3500-4000 mW) to be distributed 360 degrees around the intruder robot. The light was emitted upwards from the LED and was reflected off the inverted cone and redirected outwards away from the center of the robot.

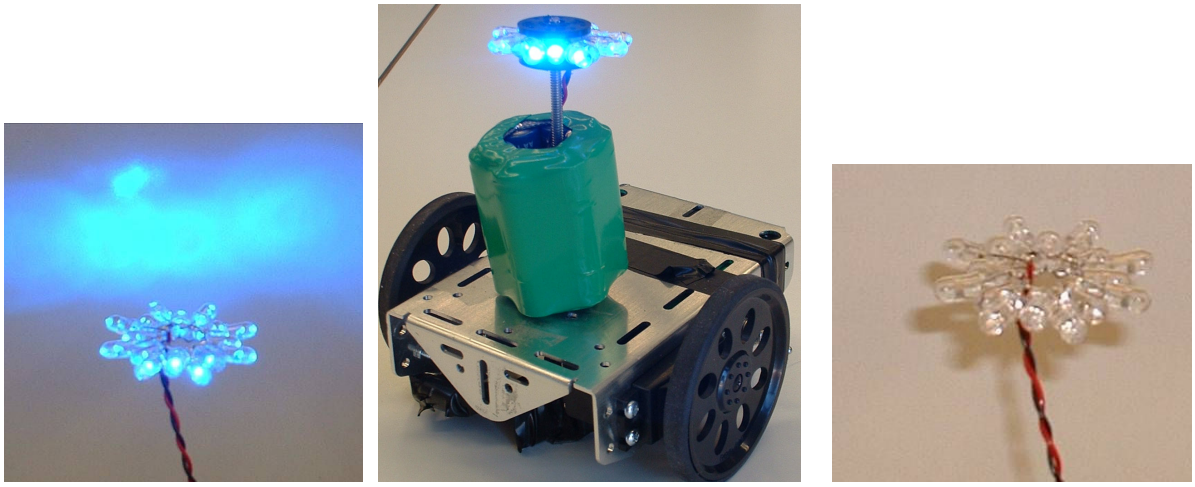


Figure 19: Intruder with LEDs

3.4 Positioning System Methods

In order for successful operation of the drone robots, a precise positioning system was required. This system was required in order to accurately command each robot to any point in a 2-D space. Each robot should be able to both accurately and precisely navigate to any given set of coordinates within a specified tolerance (tolerance and error are discussed in the analysis section). Due to the cooperative operation of the robots, they should also work with the same coordinate system so that a particular set of coordinates means the same thing for each robot. The coordinate system can also be used to set boundaries, meaning that physical walls may not be necessary. Lastly with respect to the intruder, if located by one of the drones its position can be accurately conveyed to the remainder of the drones.

Several methods for determining the position of the robots while operating in a defined area were investigated. Each method had benefits and disadvantages. Time based movement, Global reference positioning, inertial navigation, electromagnetic guidance, and several methods of odometry were all researched in the development of the drone robots. The system that was eventually chosen was a method of odometry that used an optical rotary encoder.

3.4.1 Time Based Positioning

The highest error rated method was that of time based positioning. The basic concept of this system was movement over defined periods of time at approximate velocities. This method achieves partially accurate positioning. However, basing movement on time introduces much possibility for error. Propulsion errors were large and were difficult to minimize as drive velocities varied with the type of movement, the acceleration necessary to achieve the velocity, power train status, terrain, and the kinematics coupling the system to the environment. The produced results worked, however the tolerance between each robot made the system both impractical and imprecise. Lastly, such a system has compounding positional errors which were a function of distance traveled.

3.4.2 Global Based Positioning

The system was based on measured distances to known positions and was accurate and precise however, the equipment needed to operate this method was expensive and out of our budget. The systems ability varied when taking the requested change of distance with respect to the space to measured. Most of the measured distance systems rely on some sort of electromagnetic wave transmitter and receiver system. The most popular are GPS and LORAN. These systems rely on being able to calculate the difference in time it took the radio wave which traveled at a constant speed to arrive at a location from multiple transmitters. Accuracy was largely a function of the frequency used as one wavelength was ideal. With GPS for example, a position could be obtained within a few centimeters with respect to the whole world. In the case of the drone robots, a few centimeters was a large area as the whole system wasn't very big. A positioning system that used distance measuring would require short wavelengths; these wavelengths were so short that the cost of implementing such systems would be prohibitive.

3.4.3 Inertial Navigation

Inertial navigation determines an object's placement in a defined space by measuring accelerations in various axes and integrating these measurements with respect to time. This system can be very precise but not accurate as it was subject to drift. It required periodic updates of its location by a global measurement system. These systems also have the same price short comings as the GPS style systems in that more accurate systems are expensive.

3.4.4 Odometry

This left the final method of positioning an object in a defined area, odometry. The method utilized a system of sensors which estimated the distance traveled by a wheeled or track-driven robot. The most popular odometry method was measurement of the angular displacement of the wheels or drive train. This system was particularly accurate when the interaction between the drive train and ground was ideal. This meant no slippage; one rotation of a wheel with the diameter of one unit traveled exactly Pi units with respect to ground. Vehicles driven with tracks or tank-steer had a hard time maintaining accuracy after a series of movements due to friction and slippage that could not be modeled.

Tread Counter

The initial odometry method proposed involved the use of optical encoders attached to the spur gears of the tread drive system. After initial experimentation, it was determined that this method introduced a highly significant amount of error due to the tread slippage issues discussed above. These errors resulted in the abandonment of this positioning method.

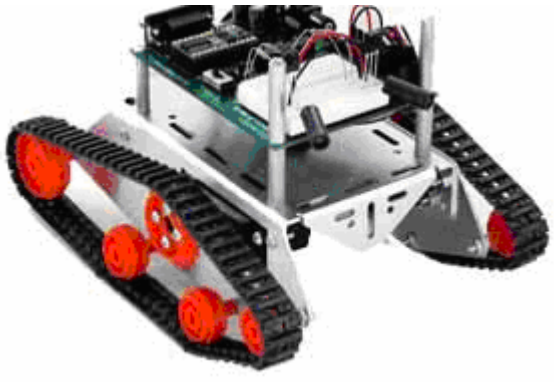


Figure 20: Boe Bot Chassis with Tank Treads

Mouse (surface)-Based Odometry

In an effort to maintain the implementation of the tread-drive system for the robots, methods of using optical and ball computer mice to determine position were thoroughly investigated. It was hypothesized that this would eliminate the slippage problem by removing the tire/surface interaction from the equation. This method required that onboard sensors directly evaluate the ground, surface, and identify vehicle movement by detecting the changes in the surface. This method was widely used in the macro field with ground firing radar which measures Doppler shift to determine movement. In the micro field of the drone robots, the preference was the use of optical systems. These systems detected the change in surface texture or movement of surface anomalies with respect to their field of view. This method was relatively inexpensive and was thought to have a supposedly high level of precision and accuracy on surfaces.

The proposed operation of the mouse (surface)-based odometry system was as follows; the counts in the x and y axis were to be tabulated for each sensor. The x values were ignored as they were only indicating slippage in a direction perpendicular to direction of travel. These values were used to detect the lateral movement such as an outside force acting on the drone. The position of each drone was determined by analysis of the separate y axis counts for the left and right sides. When the count was equal to the right count the drone was moving in a straight line. If the left count was higher the drone is turning right; if the right count was higher the drone was turning left.

The initial devices used for this method were optical PS/2 computer mice. Preliminary testing showed that these devices would be successful at completing this task. Communication between the mouse and the robot's Cubloc® processor was established. Using these mice required the addition of a PIC microcontroller chip. However, as programming was further developed, several major sources of error were discovered. The optical mice calculated position by integrating an observed velocity which lead to a large number of rounding errors. In further pursuance of this method, standard PS/2 ball mice were investigated. Ball mice were promising at first as they measured displacement instead of velocity; however they still had poor repeatability. This was caused by the fact that the values drifted when the mouse was idle. When data of positioning was compiled over time, this produced errors, greater than one foot in

magnitude. When used in a small operating environment (six by nine feet), this large magnitude was highly unacceptable, resulting in the abandonment of this positioning concept.

Wheel-based Odometry (Method Used)

The positioning system that was chosen for this project was a method of wheel-based odometry. Often used in robotic applications, this method was overlooked at first due to the fact that it would not work with the tread-drive system. The decision was made to switch to wheels and use optical rotary encoders to measure the number of rotations that the wheels traveled.

In this method, a sensor was used in collaboration with an encoder pattern that was glued to a gear. The encoder pattern is made up of black and white wedges that are equally spaced around the gear. A sensor adds to a running total every time a black wedge passes. These two totals are used in the following algorithm to resolve the current heading and position of the drone.

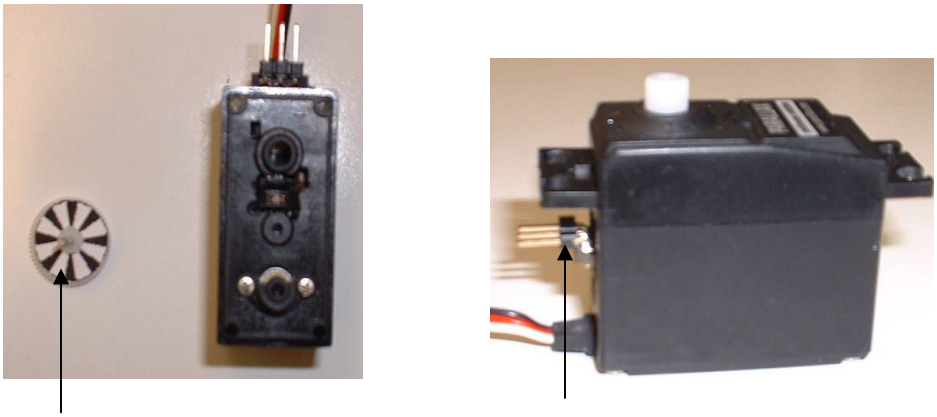


Figure 21: Wheel servo with optical encoder fixed to the inside of the servo

$$L_{DISTANCE} = L_{COUNTTOTAL} * N_{INCHESPERCOUNT}$$

$$R_{DISTANCE} = R_{COUNTiTOTAL} * N_{INCHESPERCOUNT}$$

$$If: L_{DISTANCE} = R_{DISTANCE}$$

$$X_{POSITION} = X_{ORIGINAL} + L_{DISTANCE} * \cos(\theta_{ORIGINAL})$$

$$Y_{POSITION} = Y_{ORIGINAL} + R_{DISTANCE} * \sin(\theta_{ORIGINAL})$$

$$Else:$$

$$\theta_{NEW} = \frac{(R_{COUNT} - L_{COUNT}) * N_{INCHESPERCOUNT}}{D_{WIDTHOFVEHICLE}} + \theta_{ORIGINAL}$$

$$A_{DISTANCE} = \frac{D_{WIDTHOFVEHICLE} * (R_{DISTANCE} + L_{DISTANCE})}{2 * (R_{DISTANCE} - L_{DISTANCE})} /$$

$$X_{POSITION} = X_{ORIGINAL} + A_{DISTANCE} * (\sin(\theta_{NEW}) - \sin(\theta_{ORIGINAL}))$$

$$Y_{POSITION} = Y_{ORIGINAL} + A_{DISTANCE} * (\cos(\theta_{NEW}) - \cos(\theta_{ORIGINAL}))$$

3.5 Main circuit boards and components

The main board of each robot serves as back plane for all of the components. All connections between parts are made on the surface of the board. Numerous header connectors were attached for connecting various sensors and external circuitry. The main board in actuality was a Cubloc™® CB280 prototyping board adapted for use on these robots. The board was made from double-sided copper clad fiberglass. All components are thru-hole mounted meaning fast and easy repair due to accessibility. The board, as designed for prototyping, has ample space for component placement as well as easy access to all the IO lines of the processor. Holes were drilled in the board to ease in the mounting of large components like the charging port. Components were fastened to the board by means of soldering; most components are located on the outer perimeter to free space in the middle for future component additions. The main board, when mounted to the metal chassis of the robot, served as the backbone of each robot.

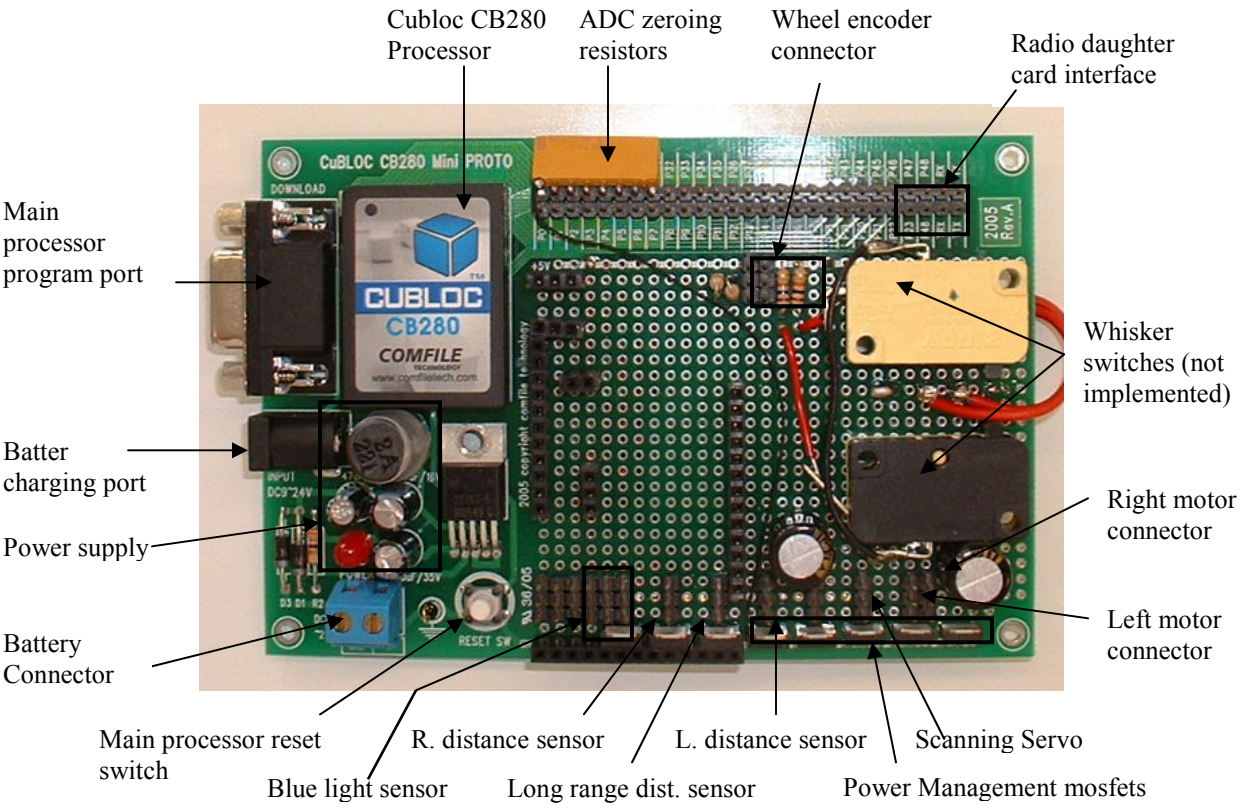


Figure 22: Main Circuit Board

The power supply for each robot was located on the main board. It was based on the LM2576T switching regulator. This device regulates the high non-stable voltage provided by the battery packs into a stable five volts which were used for all voltage-sensitive components. Due to the switching nature of the regulator, there was very little energy lost converting from the higher battery voltage down to five volts. However not all components operate solely on the five volt bus. The drive motors, used for propulsion, used the raw voltage of the batteries. This means that as the battery packs discharge, the motors operated slower. This method was chosen to isolate the sensitive electronics from the back EMF created by the operation of the motors. The radios used on these robots also operate on a lower voltage then the five volt main bus. The three volts necessary for the operational radios was provided by a low current regulator pulling power from the 5 volt bus. The power supply system for these robots was very efficient, meaning longer run times on the batteries before recharging.

To further decrease the power consumption of each drone robot; all sub systems can be powered down either individually or collectively. Mosfet switches are used to control which subsystems are active. An adaptive power scheme was devised to maximize the run time of the fleet of robots. Each drone is assigned a different power usage layout based on what its task in the fleet is. For instance if the drone is in sleep mode, only the main processor and radio is powered; while in active mode all sensors and motors can be energized.

3.6 Programming

Programming was a crucial part of this project as it provided the guidance given to the robots. This part of the project required the user to understand the computer science field. Programming was the second most important step in the project (the first being the design and manufacturing of the robots). This project would be unaffective if there was no direction given to the robots to tell them how to interact. Looking into other literary sources concerning robotic positioning and models, this project created an advanced track and trap method the robots followed. All Algorithms and code were broken into two parts, the code which is sent to each drone and the code running of the base station.

3.6.1 Interactive Graphical User-Interface (GUI)

The graphical user-interface (GUI) was essential to this project as it allows future teams to run the same code without an understanding of how it works. The GUI must be intuitive for users who will not receive a great deal of training. Some training will be available from past students who worked on the project, however in the future it may be more difficult to find. The GUI has many requirements, many of which are very time consuming to follow through with.

The GUI provides a simple, user-friendly way of creating, saving, and loading different simulation scenarios. A scenario consists of a series of robots, a user-defined environment, static but randomly placed obstacles, and various robot "states." There also is a way for a user to change general preferences, add or remove robots from the software, and view robot vital statistics without looking at or modifying existing source code. Finally the GUI must show a real-time simulation view, which has pseudo-scale robot representations moving about the screen as they are actually doing it in the simulation. For each task detailed in this report, there must be a graphical counterpart to illustrate that task such that the user need not understand source code to utilize the software.

3.6.2 Development of GUI

Before a user-interface can be developed, the tasks that the GUI represented must first be developed. In this case, the GUI and the represented tasks were done in parallel. For instance, when functionality becomes available to install or remove a robot, a GUI representation of that task was developed before another task was started. The GUI part should always be done after the task has been completed. This means that if a task has become delayed, the GUI representation of that task has also become delayed.

3.6.3 GUI Obstacles

There comes a time when the GUI cannot be developed until another part of the project was also developed. For example, until the position system was tested and working, it was nearly impossible to test with any certainty the quality of the simulation viewer on the master software. This module was tested by feeding it sample data, however many variables such as refresh speed cannot be significantly tested until position is completed.

The only difficulty was the issue with robot representation scaling on the simulation viewer module of the master software. Initially the robots were to be on a 9ft by 6ft environmental test platform. With the resolution of a standard computer monitor, both the test platform, and the robots could be drawn to scale on-screen while still showing the heading of each robot. When the scenario changed to placing robots on a 30ft by 30ft environment, the drawing also changed on-screen. When drawing a large environment to scale, the 8-inch robots become 1 pixel or less! One pixel was not enough detail for the user to accurately see what is happening, nor was it enough space for the user to click on to view details. To fix this problem, the robots have a minimum size. When environments grow large enough, the robot was scaled to that minimum size. This means for large environments, the robots will not be drawn to scale on the screen. This may give users some confusion; however it was the only alternative to presenting the user with unusable software.

3.6.4 Present Programming

Currently there are many modules that have been developed and tested with a graphical interface counterpart. The user can currently create a new simulation, selecting any of the installed robots. The user can assign each robot a "state" when starting the simulation. These states included: active; moving, sensing and looking for the intruder; paused; stopped, but sensing and looking for the intruder or sleeping, not moving, not sensing, waiting for a request from the master to become active. The new simulation dialog also provided a graphical view of each robots communications test. Within this dialog, a user can define a custom environment size, and provide specific pre-simulation commands. These commands included stationing robots at various key points within the environment, and placing them at specific headings.

The user can also view a list of installed robots, add robots, and remove robots graphically. In addition, each robot can be viewed using the robot health viewer, a module designed to show vital statistics about each robot including voltages, sensor outputs, and communication status. The simulation viewer was on hold due to the positioning system. The load and save simulation module was complete, however it needed a graphical interface to represent it. The general preferences module also needed a graphical interface to represent it. Once these modules have been added, and the positioning system has been finalized the graphical interface will be close to completed.

4. RESULTS

Multiple tests needed to be conducted in order to test each sensor individually as well as to test multiple sensors at once. In addition to the sensors being tested, tests were run with the introduction of obstacles to test if the robots could handle stationary objects blocking their path. Light intensity from the intruder was varied to determine how effective the robots would react to a limited light source.

4.1 Program: 'RoboSim V2.0'

The GUI was completed allowing a user with no previous knowledge of the software, the ability to sit down with little instruction, and run the program. Many hours were spent creating the background coding to run the program. This background coding can be looked at in Appendix E, as there are numerous pages of programming that were developed to create this user friendly interface. Finalized, the following program was labeled "RoboSim V 2.0"

Opening the program, the user will start off on a blank screen, as seen below. This is the main screen that opened up when the program was run. The screenshot below provides a visual to help the user understand what they are looking at.

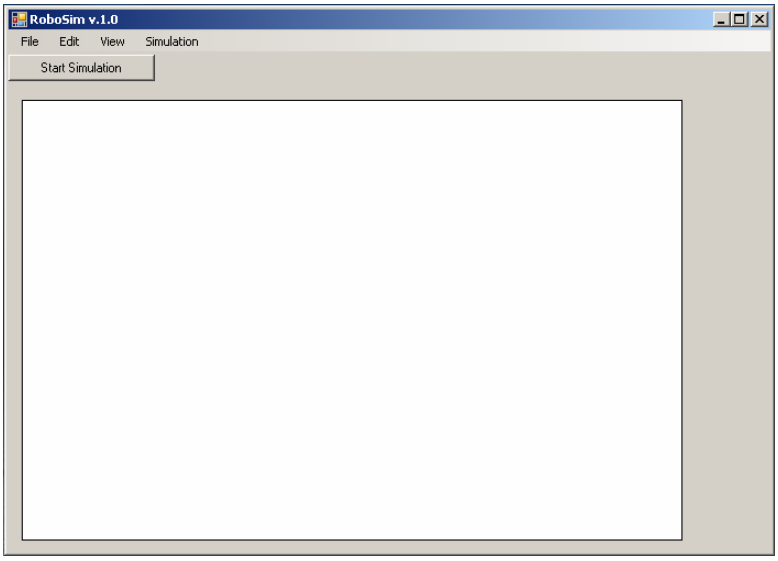


Figure 23: RoboSim's Main Screen

Once this screen is opened, we created a 'new simulation'. This function allowed us to start a new project, giving the robots new positions and area to test in. The ability to 'load simulation' is also available from this menu. If chosen, one is able to load previous settings of

test conducted in the past. The screenshot below provides visual aid on how to reach these two options.

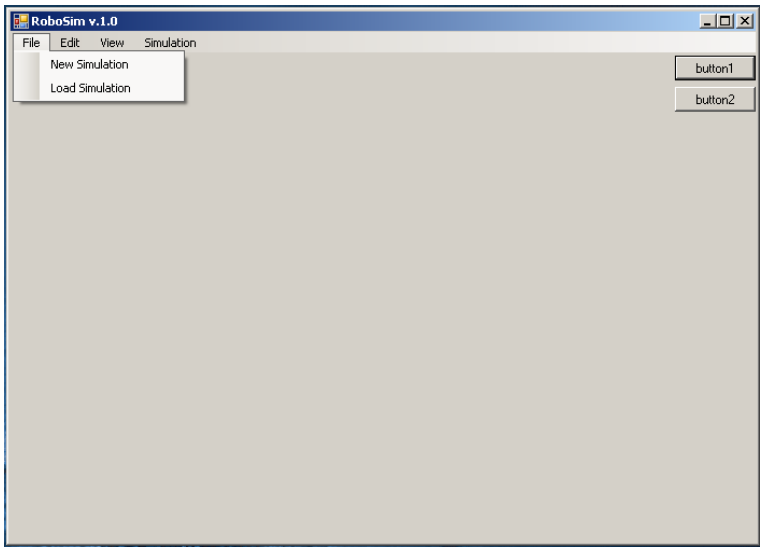


Figure 24: RoboSim's New Simulation Drop Down Menu

The next step was clicking on the 'Simulation' tab. A drop down menu brought up three options, 'Communication', 'Robot Health', and 'Simulations Specs.' When 'Communication' was clicked it brought the user to a scenario set-up page which will be discussed in a moment. The 'Robot Health' and 'Simulation Specs' buttons were not clicked as there was no scenario currently programmed into the software. Below a screenshot provides another visual aid of the step explained above.

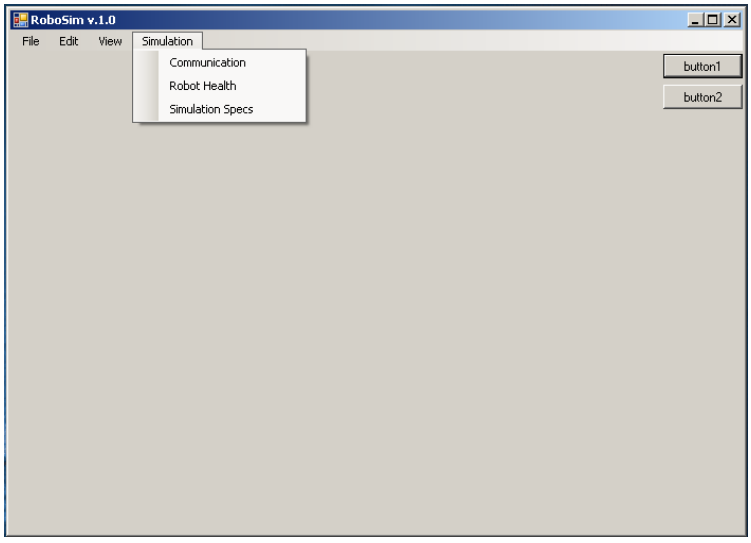


Figure 25: RoboSim's Simulation Drop Down Menu

After clicking the ‘Communications’ link, we were brought to the ‘Robots’ tab. A number of important variables were displayed on this screen. First, we checked off the number of robots used for this particular test. The number of checked robots could vary depending on the scale of the test. Highlighting an individual robot, the program displayed the ‘Robot Status’ box whether the robot was ‘active, paused, or sleeping.’ Active meant the robot was moving in an attempt to find the intruder; paused meant the robot was stopped somewhere on the test platform; and sleeping meant the robot was charging in the chute. The box to the right labeled ‘Team Status’ displayed a list of robots the user selected and their statuses. When one robot was selected, individual robot data was displayed in the ‘Robot Comms’ box on the bottom right of the screen. Here you saw the robots name, ID #, Robot Comm ID #, and whether it was initialized. Dependent upon the scenario, the user had the ability to choose from 1 to 9 of the robots available. To clarify where exactly everything is on this screen, another screenshot is seen below to provide a better understanding of the directions above.

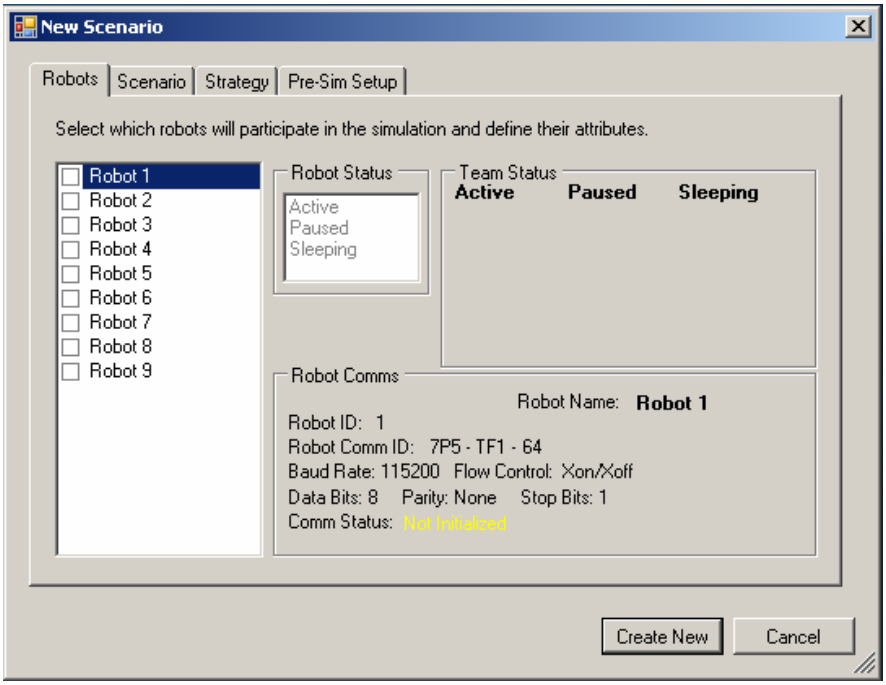


Figure 26: RoboSim's Scenario Menu

After the ‘Robots’ tab has been set up, the user moved on to the ‘Scenario’ tab. At this tab, the user was required to set the dimensions for the testing area. In the case below, a 9ft X 6ft area was chosen, as it was the size of the ETP. The user clicked on the ‘length’ and ‘width’

boxes to input the dimensions. Various dimensions can be used, depending on the specific scenario run. The screenshot below shows a visual on where to input values.

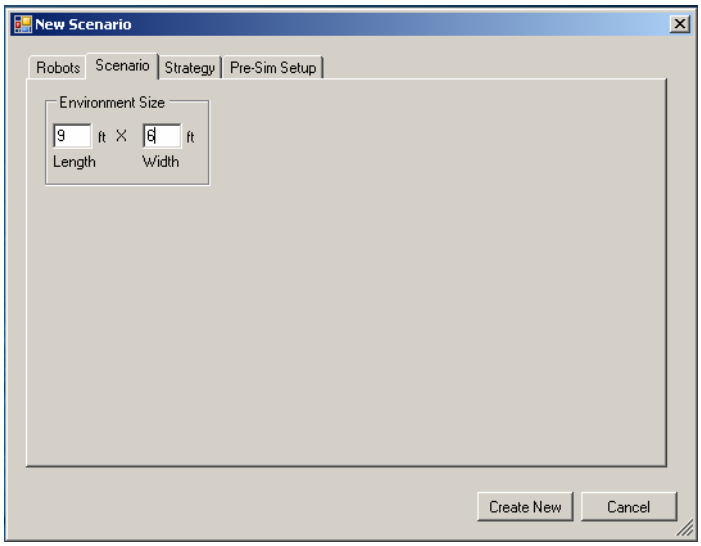


Figure 27: RoboSim's Environment Size Menu

The last stage of the program preparation required us to place the robots we chose in the area we specified. As stated in section 2.3, the robots started in the chute as that was where their global origin is found. The chute was always and will always be placed in the bottom left most portion of the designated area. The program was written with these known variables, and calculates the position of the robots off of this known origin.

Clicking on a robot in the 'Avail Robots' tab, the mouse was then moved over the white space depicting the testing area, where a blue outline of the robot appeared. Notice that the robot's size was to scale with the specified area. Moving the mouse over the white space, the robot was placed in the top right corner for demonstration purposes. The robot was only able to be placed if the icon was blue. If red in color, the robot is not in the programs boundary area. Once the robot icon was placed, one had the option to rotate the face of the robot by sliding the pointer in the 'Heading' box to a given angle. Once the robot placement and angle was designated, the next robot (if available) was chosen and the procedure was repeated for as many robots as were selected in the "Robots' tab. At any point the user is able to remove a robot by clicking on the blue robot icon and then clicking the 'Remove' button. Once all the robots were placed, the user hit the 'Create New' button, at which point the activated robots moved to their designated positions and began searching for the intruder. Below is a screenshot to help understand where the controls are for the directions stated above.

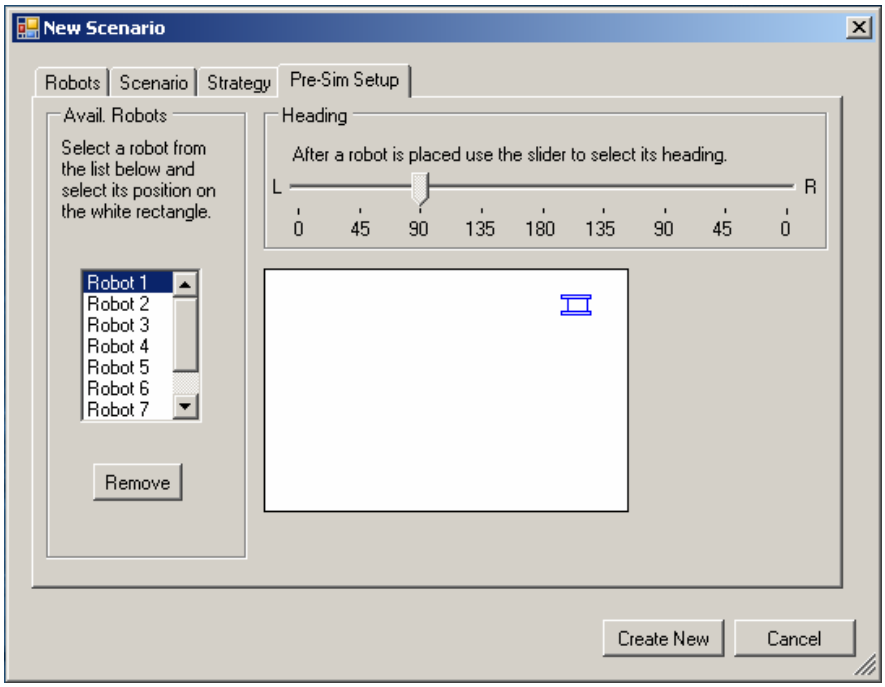


Figure 28: RoboSim's Pre-Sim Setup Menu

Once all the steps above were completed, the robots moved to their designated positions and began scanning for the intruder. While the robots ran their scenario, another section of the software allowed the user to monitor various components of the active robots. Once the user clicked 'Create New', the robots began to move and the computer screen flashed back to the main page. At this point, the user clicked on 'Communications→Robot Health', which opened the window seen below. In this window, the user chooses a robot that they would like to see the status of. Status of the robot is checked every 700 milliseconds, which is safe to say that the robots are continuously being checked. Clicking robot 1, the user verified the robots ID in the 'Robot Identification' box. The robots communication status in the 'Robot Communication' box displayed that the 'comm status' was 'not initialized' because the robot was not in use or the radio was not communicating to the base station properly. The user was able to track the robots position at any time in the 'Robot Positioning' box. Values were presented in the 'Sensor Voltages' and 'Actuator PWM' that allowed the user to track the accuracy and power available to the robot. To exit the 'Robot Health' screen, click on the 'Close' button. Below is the screenshot to aid in following the directions above.

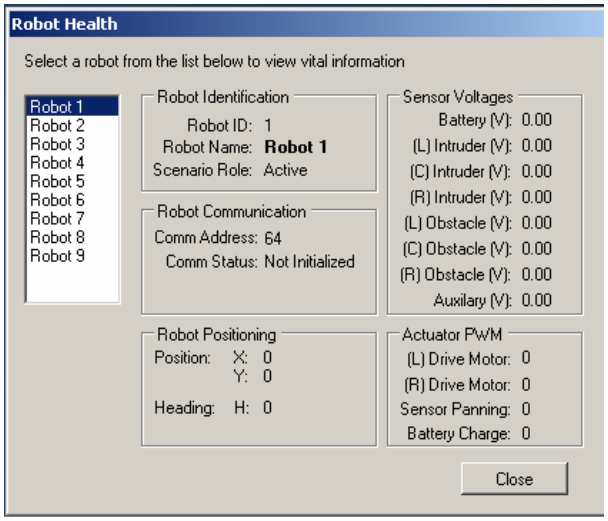


Figure 29: RoboSim's Robot Health Menu

4.2 Large Testing Environments

The initial testing took place on the ETP, and the program was debugged. Satisfied with the results the robots were taken to WPI's squash courts which provided a larger testing area. Within this area, tests were taken to determine if the maximum range values of the sensors and were correct. To verify our values a basic test was conducted with the intruder and seven of the robots.

The test started by inputting the parameters, into the 'RoboSim' program. Seven robots were used, and the chute was placed in the bottom left corner of the court which established the robots origin. All the 'pre-sim' information was uploaded into 'RoboSim.'

Figure 30: Robots in the Chute (Origin)

The robots were mapped in the program to align around the perimeter of the squash court. The perimeter of the squash court was 15ft X 30ft, but we reduced the area to a 15ft X 20ft area.

The intruder was placed in the center of the field. Once the intruder was placed, the 'Create New' button was pushed in 'RoboSim' and the robots began to move to their assigned positions.

Figure 31: Robots Moving to their Starting Positions

Once in their assigned positions the intruder's light was then flipped on and the robots began searching for the intruder.

Figure 32: Robots in Position; Awaiting the Start Command

The robots then began moving towards the intruder while the intruder was moving towards the top left corner of the picture. We noticed that the robots fluidly corrected their positions to adjust to the intruders movement.

Figure 33: Robots Initial Movement Towards the Intruder

The intruder continued to try and move through a gap between the oncoming robots, but the continual updated position of the intruder to the surrounding robots, allowed for quick corrective movements preventing the intruder from escaping.

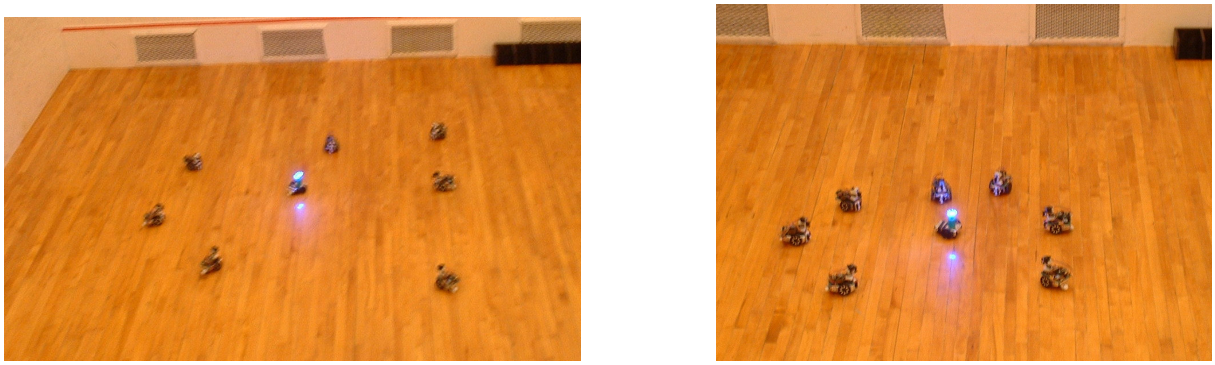


Figure 34: Robots Tighten the Gap around the Intruder

In the tests final stages, the intruder was unable to move as the other robots had successfully contained the intruder. At this point, the robots kept their assigned minimum distance of one foot from the intruder.

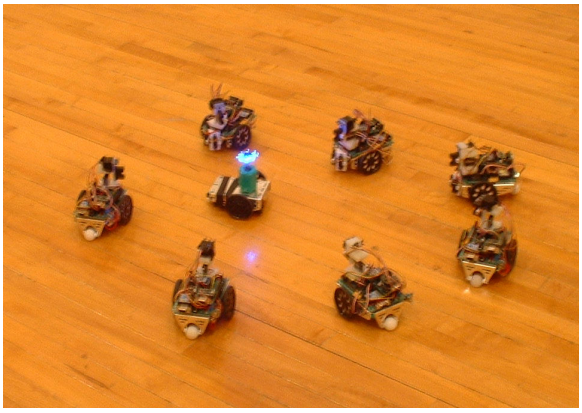


Figure 35: Contained Intruder

4.3 Sensors

The 45 degree angle mounted short distance infrared sensors worked perfectly in detecting oncoming obstacles and redirecting the robots path with plenty of space between the obstacle and the robot. This minimized the probability that the robot could get stuck or boxed into a corner, and use valuable energy to maneuver its way out. The short distance sensors accurately detected an object about 18 inches away.

The long distance infrared sensor placement proved useful as it was mounted higher than the short distance sensors and could see above a few of the obstacles that the short range sensor could not see above. Combined with the light sensors, once both sensors detect the intruder through the created algorithm, the robot accurately hone in on the intruders' position.

The light sensors played the most important role in detecting the intruder. Since the intruder's only unique variable was that it gave off light, it was important that the light sensors covered a wide area over a great distance. The light sensors discerned the difference between outside variables and what was actually the intruder; something the infrared sensors could not do alone. With a range of about 62 inches, using seven robots in a 30 foot square grid, it does not take long for one robot to spot the intruder and relay this information to the other robots to eventually contain the intruder.

4.4 Radio Communication

The only method of communication between the robots and the base station was the radios. The radios needed their own processor that was used to delegate a number of functions within the program. With nine robots working together, there was a lot of "chatter". The processor was able to filter this chatter to specific robots, relaying information about the position of the intruder which can be transmitted to other robots to quicken the containment process. The algorithm created for this project lets the processor know what information was critical and which robot needed to process the information.

The radio processor used the Cubloc® base program which was written for this project. The software used has limited processing capability, but enough for the processing required for this project. If more sensory equipment was added to the robots or the number of robots was increased, it may be necessary to improve the quality of radio communication.

5. ANALYSIS

The development of the intrusion detection network using mobile sensor arrays involved the integration of multiple subsystems into a system capable of dynamically determining the position of an intruding or contaminating object with known quantifiable characteristics. Once requirements were defined, the initial base hardware was acquired and development of the required systems was completed.

The subsystems and components that were integrated into the system were done so after several iterations and changes. A key component of the system was the method of determining the global positions of each individual drone robot. The odometry method that was finally chosen included the use of optical rotary encoders embedded in the servo-drive motors of the drones. Compared to the other methods investigated, this system provided precision measurements of the distance traveled by each drive wheel, allowing sufficient determination of the robot's heading and position in the given environment. A benefit of knowing the accurate position of each robot includes the ability for the base station to direct other drones efficiently so as not to cover the same terrain multiple unnecessary times and to prevent situations where drones might interfere with each others' movement.

The final arrangement of the detection sensors resulted in the system being capable of effectively discriminating an intruder with a known "contaminant signature" from other drone robots and immobile objects such as walls and other obstacles. Based on the measurements obtained by a single robot, the general location of an intruder is distributed to a given number of active drones which are sent to further define the exact location of the intruder. This allowed the system to use a minimum amount of movement and system resources necessary contain the intruder. The five degree angle that was chosen for the blue light sensors was compatible with the code developed, allowing the robot to center its sensor head on the intruder and make intelligent movements based upon both the angular orientation of the intruder and the algorithms present in the robots control programs. The development of a short range object detection method began with the inclusion of both physical contact whiskers and short range infrared distance sensors. The decision to eliminate the whiskers proved to be acceptable due to the angular orientation of the two IR distance sensors being capable of preventing unwanted contact.

The finalized intruding contaminating source that was used to test the system is operated by human-interface and is completely controllable. There are several advantages and disadvantages to this approach. The use of an intruder controlled by hand is beneficial in that it allows for testing and manipulation of the device by a wireless radio controller. Thus, it is easier to test a variety of movement tactics and methods in an attempt to "fool" the sensor network. However, a downside of this method is that the exact movements and positions of the device are not known (unless physically measured), disallowing the direct evaluation of the detection system's accuracy. One of the Cubloc based robots could have been outfitted with blue light emitters and used as an intruder. This was not done due to the fact that further programming of each individual scenario would be required. The main benefit of the human-interfaced intruder is that it allows for random control to test every aspect of the system.

Development of programming and algorithms for control of the drone robots also required knowledge of the capabilities of the sensors and drive units. Often times, sensor components and other hardware were modified due to limitations in programming. The final program that was developed successfully incorporates the sensor inputs with control algorithms to track and detain the intruder. The PC-based graphic user interface allows for "on-the-fly" notifications and scenario updates, proving to be very useful in the operation of the system. The central hub server programming has the ability to intelligently delegate tasks and functions assigned by the main server code to the drone robots.

The operation of the radio communication programming proved to have minor flaws that resulted in disrupted communications between the central hub and the drone robots. Countless hours were spent developing and troubleshooting the radio communications problems, but time did not permit for all the bugs to be fixed. It is recommended that future MQP groups include students majoring in computer science and electrical engineering. The overall functionality of the entire mobile sensor network is successful except for the issues discussed regarding the radio communications programming.

6. CONCLUSION

The primary goal of this project was to develop a system that could effectively use sensor measurements, wireless communication, and a series of fully developed algorithms to expeditiously detect, track and detain an intruder. Initially, this project presented the challenge of learning programming as well as some electrical engineering skills. A user friendly programming system was chosen, Cubloc ® Technologies, which was used to aid us in developing an advanced program that allowed us to give orders to multiple robots over one network. Initial iterations showed that there were vast numbers of possible combinations of types of sensors, styles of movement, and programming techniques that could have been developed to effectively complete the desired tasks. We maximized our budget to acquire sensors, processors, and position encoders, which provided the robots with sensors capable of detecting an intruder up to five feet away. Since the base circuit board can be modified, there was no limit to the type of program or sensory components that we used. We took full advantage of the circuit boards capability and produced the most effective robot possible for the scenarios we tested.

Many hours were invested in determining an accurate method of position tracking. The focus on tracking position was crucial, as it provided the data that was inputted into the program 'RoboSim'. If robot tracking produced error, than the information transmitted from the robots to the base station would be inaccurate and the intruder would mostly likely not be contained. Tracking an intruder in a 20ft X 20ft area was completely accurate and effective. If this project were to be implemented on a larger scale, systems already in place (GPS) could be used to track position. The network that the robots run on made the best use of the sensors and processors available to our project. In the future, time can be spent on the improvement, rather than the design of this network, as the need for something new in the field of autonomous robots is in high demand. Overall, the design and integration of the various subsystems resulted in a state of the art intrusion detection system capable of tracking and surrounding an intruder.

7. RECOMMENDATIONS

A number of iterations for the light sensors, distance sensors, position tracking, style of movement, and programming were evaluated and the current design best suited our MQP's environment. Now that base of the robots and programming have been created, future project groups can focus more time how to make the program better, rather than how to make the program function. Numerous hours were spent on creating an in-depth program, but as in any project, the more one analyzes situations, the more opportunities they find for improvement.

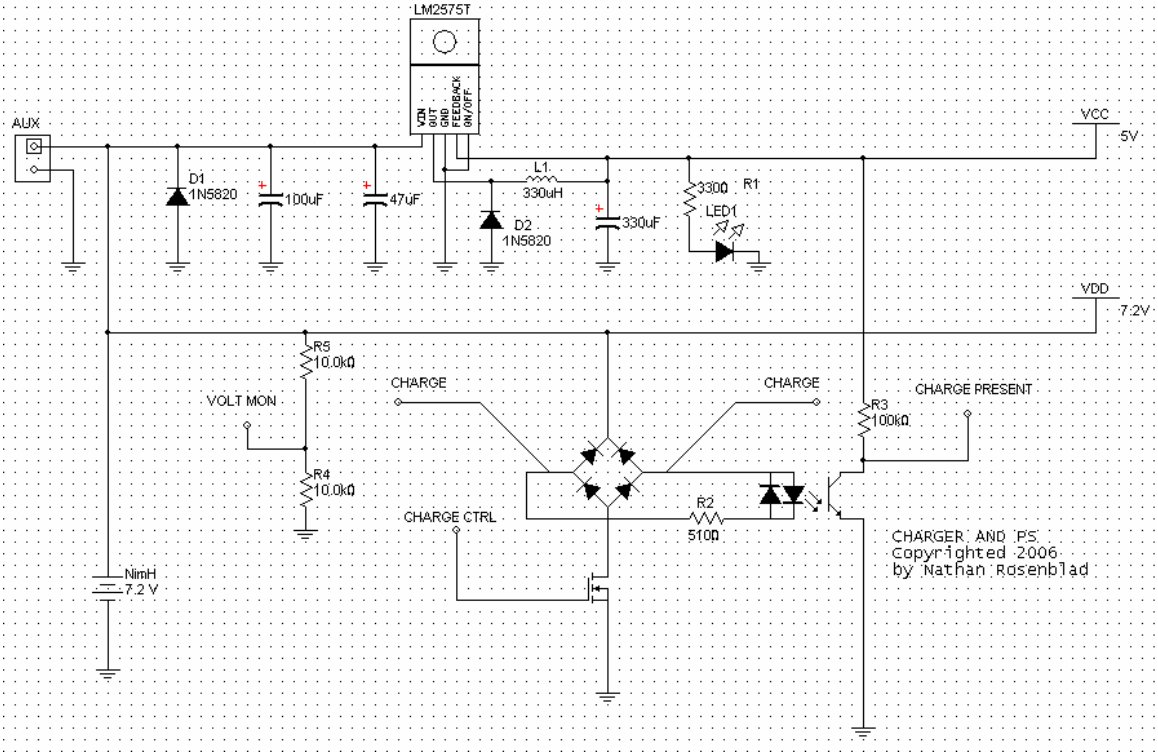
The programming for this project has a major significance in this project as it gives the robots direction and a mission. Since the group members were not Computer Science majors, it took a substantial amount of time to learn and write code that would meet the specifications of the MQP project. In the future it would be extremely beneficial to have an electrical as well as computer science background, in order to maximize the work output efficiency.

References

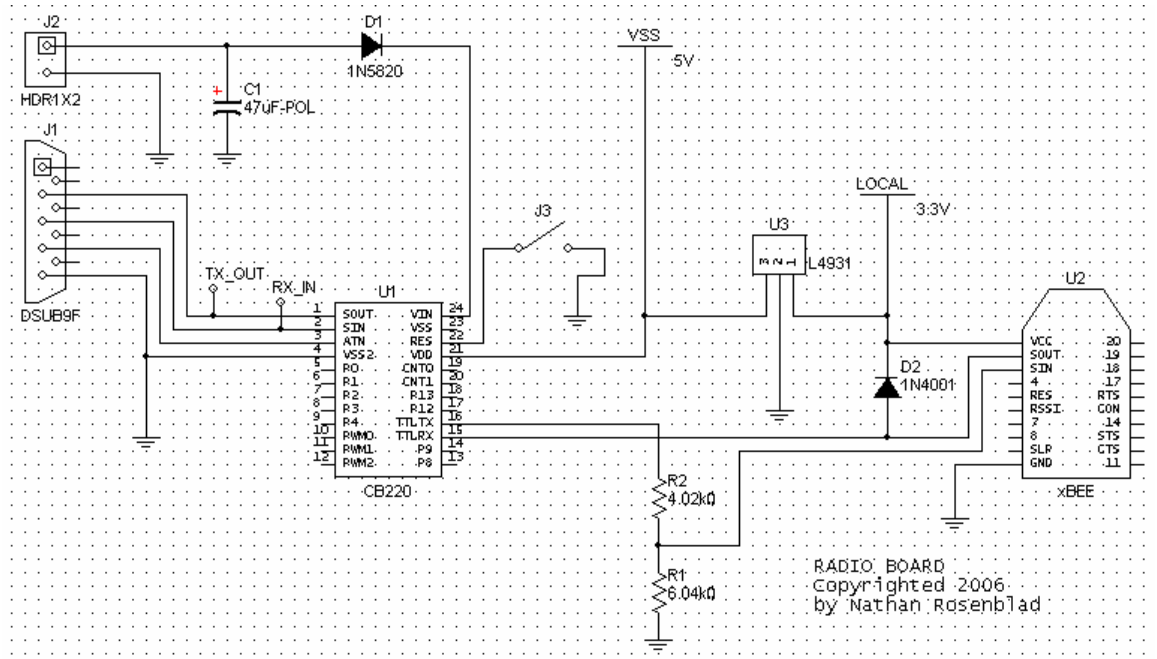
1. COMFILE Technology, Version 2.0; *Cubloc User's Manual*; COMFILE Technology ©; Foster City, CA; 2006
2. Holland, John M.; *Designing Autonomous Mobile Robots*; Newnes Publications, 2004
3. Meystel, A.; *Autonomous Mobile Robots*; World Scientific Publishing, 1991
4. Military '*Predator'* UAV: http://www.fas.org/irp/program/collect/predator.htm, January 30th, 2007
5. Multiple Autonomous Robots: http://www.cis.upenn.edu/mars/ , February 1st 2007
6. 'TSL257 High-Sensitivity Light-to-Voltage Converter Data Sheet', TAOS Inc., Copyright 2001

Appendix A: Robot Component Schematics

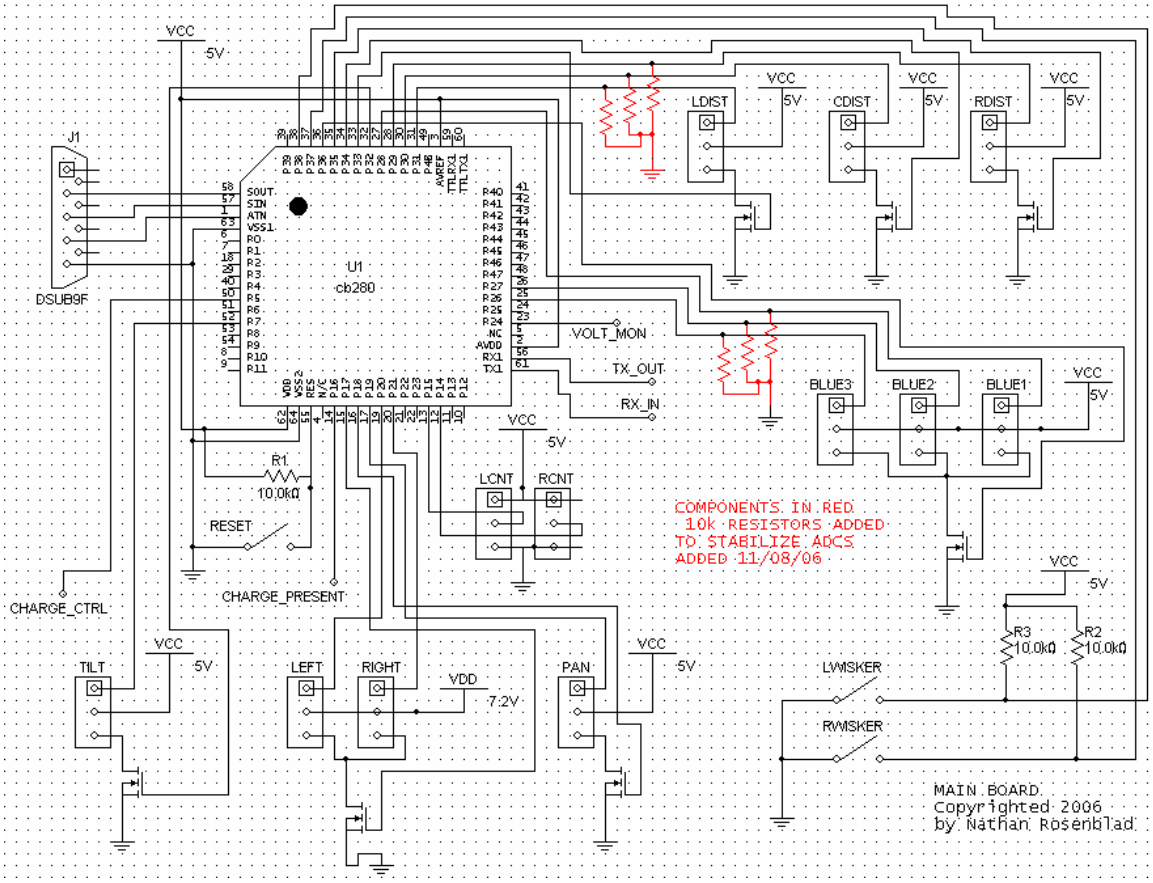
DRONE ROBOT POWER REGULATOR & CHARGING SYSTEM



DRONE RADIO TRANSPONDER SUB BOARD



MAIN BOARD & SENSOR ACTUATOR CONNECTIONS


VCC
5V
J1
DSUB9F
U1
cb280
SOUT
SIN
ATN
VSS1
RO
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R40
R41
R42
R43
R44
R45
R46
R47
R27
R26
R25
R24
NC
AVDD
RX1
TX1
VOLT_MON
TX_OUT
RX_IN
LDIST
CDIST
RDIST
BLUE3
BLUE2
BLUE1
LCNT
RCNT
R1
10.0kΩ
RESET
CHARGE_CTRL
CHARGE_PRESENT
TILT
LEFT
RIGHT
VDD
7.2V
PAN
LWISKER
RWISKER
R3
10.0kΩ
R2
10.0kΩ
COMPONENTS IN RED
10k RESISTORS ADDED
TO STABILIZE ADCS
ADDED 11/08/06
MAIN BOARD
Copyrighted 2006
by Nathan Rosenblad

Appendix B: Robot Movement Equations

Algorithms

$If: R_{COUNT} > L_{COUNT}: Then$

$$\frac{\Delta_{COUNT}}{R_{WIDTH}} = \frac{L_{COUNT}}{R_{INSIDE}}$$

$$\frac{R_{COUNT} + L_{COUNT}}{2} = A_{COUNT} = (R_{INSIDE} + \frac{1}{2} R_{WIDTH}) * \Delta\theta$$

$EndIf$

$If: R_{COUNT} < L_{COUNT}: Then$

$$\frac{\Delta_{COUNT}}{R_{WIDTH}} = \frac{R_{COUNT}}{R_{INSIDE}}$$

$$\frac{R_{COUNT} + L_{COUNT}}{2} = A_{COUNT} = (R_{INSIDE} + \frac{1}{2} R_{WIDTH}) * (-\Delta\theta)$$

$EndIf$

$$\theta_{HEADING\_CURRENT} = \theta_{HEADING\_PREVIOUS} + \Delta\theta$$

$$\Delta Y = (R_{INSIDE} + \frac{1}{2} R_{WIDTH}) * \sin(\Delta\theta)$$

$$\Delta X = (R_{INSIDE} + \frac{1}{2} R_{WIDTH}) * \cos(\Delta\theta)$$

$$Y_{CURRENT} = Y_{PREVIOUS} + \Delta Y$$

$$X_{CURRENT} = X_{PREVIOUS} + \Delta X$$

$ElseIf:$

$$\frac{R_{COUNT} + L_{COUNT}}{2} = A_{COUNT}$$

$$\theta_{HEADING\_CURRENT} = \theta_{HEADING\_PREVIOUS}$$

$$\Delta Y = (A_{COUNT}) * \sin(\theta_{HEADING\_CURRENT})$$

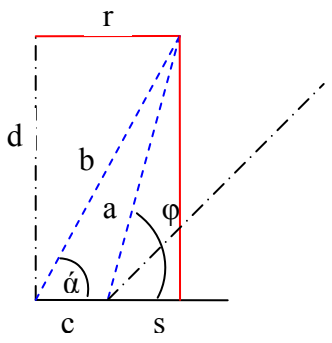
$$\Delta Y = (A_{COUNT}) * \cos(\theta_{HEADING\_CURRENT})$$

$$Y_{CURRENT} = Y_{PREVIOUS} + \Delta Y$$

$$X_{CURRENT} = X_{PREVIOUS} + \Delta X$$

$EndElseIf$

Appendix C: Full Calculations for Sensor Head Field of View



$d := 4in$

$\Theta_2 := 6deg$ $\Theta_2 = \frac{\Theta}{2}$

$r := \tan(\Theta_2) \cdot d$ $r = 0.035ft$

$\alpha := 90deg - \Theta_2$

$b := \sqrt{d^2 + r^2}$ $b = 0.335ft$

$c := \frac{.4ft}{12}$ $c = 0.033ft$

$a := \sqrt{-2 \cdot b \cdot c \cdot \cos(\alpha) + b^2 + c^2}$ $a = 4in$

$s := r - c$ $s = 0.02in$

$\phi := \operatorname{asin}\left(\frac{d}{a}\right)$

$\phi = 89.708deg$

$\theta := 90deg - \phi + \Theta_2$

$\theta = 6.292deg$

Appendix D: Initial Scanning and Panning Sensor Head Design

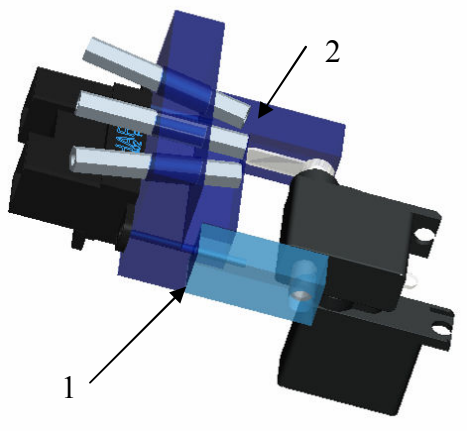


Figure 36: Panning and Scanning Sensor Head

The initial design of this panning sensor head has the ability to pan left to right, as well as scan up and down. The movement of the head is facilitated by a pair of mini servo motors as seen in the figure above. This design called for the manufacture of two separate head components (1 and 2) which were to be screwed together. This design was discarded for several reasons. The horizontal orientation of the distance sensor resulted in interference with the short range sensors. The cost of using 2 servos was prohibitive. Large moments due to the weight of the head and sensors could result in instability of the structure when panning. The largest factor in the decision to discard this design was the instability of the dual axis servo-servo mounting and the programming error that would develop due to the vertical scanning. In order to accurately determine the vertical angle at which the servo is oriented, a separate feedback device would be needed due to the fact that the sensitivity in the vertical plane is very high. A small error in the angular position of the servo shaft would result in a large error in the sensor-to-ground distance reading.

Appendix E: Main Cubloc Code

```
Const Device=cb280
Ramclear

Dim SSN As Integer 'Unique identification number used in radio communications
Dim RAW(8) As Integer 'Array of important voltages - Batt/Aux/R Blue/C Blue/L Blue/R
Dist/C Dist/L Dist
Dim PWMS(4) As Integer     'Array of PWM values - Left/Right/Pan/Batt
Dim POSITION(3) As Single          'Array of all Position Information of Robot - X/Y/Heading
Dim header(7) As Byte              'The header input data received for all radio
communications
Dim dataInput(104) As Byte 'Data that will be received for radio communications
Dim PositionMatrix(10,2) As Integer 'Matrix with Position Data
Dim LastX As Integer
Dim LastY As Integer
Dim INTRUDER(3) As Integer         'Array of all Position Information for Intruder -
X/Y/Confidence Interval(0-100)
Dim SPWM As Integer          'PWM For Panning Head Servo

Set Debug Off

'This is the MAIN function block, only the function calls in HERE will be executed

initNavHardware
initRadio
enableSensors
POSITION(0)=0
POSITION(1)=0
POSITION(2)=1.57096
LastX=0
LastY=0
Do
        updateRawVoltages
        If In(48)=1 Then
                ackMsg
                receiveRadioComm
        Endif
Loop

'End MAIN function block

End

Sub initNavHardware()
```

```
'This sub initializes all navigation hardware
	Set Count0 On 				'INITIALIZE SECOND
HARDWARE COUNTER
	Input 14 				'SET PORT 14 (RIGHT
COUNTER) TO INPUT
	Input 15 				'SET PORT 15 (LEFT
COUNTER) TO INPUT
	Countreset 0 			'RESET RIGHT COUNTER
	Countreset 1 			'RESET LEFT COUNTER
	Low 19 				'Turn off Panning Motor
	Low 20 				'Turn off Left Drive Motor
	Low 21 				'Turn off Right Drive Motor
	Pwm 4,3500,65535 		'SET PWM Left Drive TO MOTOR
NEUTRAL
	Pwm 5,3500,65535 		'SET PWM Right Drive TO MOTOR
NEUTRAL
	High 17 				'TURN ON MOTOR
CURRENT
	Input 48 				'SET PORT 48 (Radio Proc
Flag) to INPUT
	Output 47 			'SET PORT 47 (Radio Response
Flag) to OUTPUT
	Low 47 				'TURN OFF "TURN-OFF-
PIN"
End Sub

Sub delegatePosMatrix()
	Dim i As Integer
	Dim myDistance As Single
	For i=0 To 9
		myDistance=Sqr((PositionMatrix(i,0)-LastX)^2+(PositionMatrix(i,1)-LastY)^2)
		Debug Dec i,Cr

		navGoToCoord PositionMatrix(i,0),PositionMatrix(i,1),myDistance
		'If In(48)=1 Then
	'		ackMsg
	'		If header(2)=1 Or header(2)=3 Then
	'			i=9
	'		Else
	'			receiveRadioComm
	'		Endif
	'	Endif
	Next
	Pwm 4,3500,65535
	Pwm 5,3500,65535
	'receiveRadioComm
```

```
End Sub

Sub navGetPosition()
        Dim leftDistTrav As Single                  'DISTANCE TRAVELED OF LEFT TIRE
        Dim rghtDistTrav As Single                  'DISTANCE TRAVELED OF RIGHT TIRE
        Dim radius As Single                            'RESULTING RADIUS
        Dim oldHeading As Single
        Dim BOEWIDTH As Single
        Dim IPC As Single

        IPC=0.022776
        BOEWIDTH=4.304                                          'DISTANCE MEASURED
BETWEEN TIRES (INCHES)
        leftDistTrav = Count(1)*IPC  'ACTUAL TRAVELED DISTANCE OF LEFT TIRE
        rghtDistTrav = Count(0)*IPC'ACTUAL TRAVELED DISTANCE OF RIGHT TIRE

        If leftDistTrav = rghtDistTrav Then   'If both travel the same distance, it went straight
                POSITION(1) = POSITION(1) + leftDistTrav*(Sin(POSITION(2)))

                POSITION(0) = POSITION(0) + leftDistTrav*(Cos(POSITION(2)))
        Else
                oldHeading=POSITION(2)
                POSITION(2) = (rghtDistTrav-leftDistTrav)/BOEWIDTH + POSITION(2)
                                                'ACTUAL ANGLE
                radius = (BOEWIDTH*(rghtDistTrav+leftDistTrav))/(2*(rghtDistTrav-
leftDistTrav))           'ACTUAL RADIUS OF ARC
                POSITION(1) = POSITION(1) - radius*(Cos(POSITION(2))-Cos(oldHeading))
                                'ACTUAL Y CORDINATE
                POSITION(0) = POSITION(0) + radius*(Sin(POSITION(2))-Sin(oldHeading))
                                'ACTUAL X CORDINATE
        Endif
        Countreset 0
        Countreset 1
End Sub

Sub navGoToCoord(nextX As Integer, nextY As Integer, pntDistance As Single)
'This sub takes the X,Y cords of the next point it is trying to go to, as well as the distance from
the previous point, to the point it is trying to go to.
'The sub makes changes to the robots position until it is close enough to the given position to get
another command.
        Dim distFromGoal As Single
        Dim xStraight As Single
        Dim yStraight As Single
        Dim errorStraight As Single
        Dim xTurnGuess As Single
        Dim yTurnGuess As Single
```

```
        Dim errorGuess As Single
        Dim angErrorPerc As Single
  Dim disErrorPerc As Single

        distFromGoal=Sqr((nextX-POSITION(0))^2 + (nextY-POSITION(1))^2)
        Debug "Going To: ",Dec nextX, " ", Dec nextY,Cr

        Do While (distFromGoal > 0.75)
                xStraight=distFromGoal*Cos(POSITION(2)) + POSITION(0)       'X Coord if
robot went straight for distance-to-goal inches
                yStraight=distFromGoal*Sin(POSITION(2)) + POSITION(1) 'Y Coord if robot
went straight for distance-to-goal inches
                errorStraight=Sqr((nextX-xStraight)^2 + (nextY-yStraight)^2) 'How far off is that
hypothetical point?

                angErrorPerc=errorStraight/distFromGoal     'Distance from target point as a
percentage of distance to travel ideally
                disErrorPerc=distFromGoal/pntDistance                'Distance from target as a
percentage of the distance from last registered point (progress)
        'Debug Float xStraight," ",Float yStraight," ",Float errorStraight,Cr
                If  (3.0*angErrorPerc) > (disErrorPerc) Then
                'The error in angle is worse than the error in distance
                        xTurnGuess=distFromGoal*Cos(POSITION(2)+0.001) + POSITION(0)
                        yTurnGuess=distFromGoal*Sin(POSITION(2)+0.001) + POSITION(1)
                        errorGuess=Sqr((nextX-xTurnGuess)^2+(nextY-yTurnGuess)^2)
                        If (errorGuess > errorStraight) Then
                                'Guessed Correction got worse, Turn Right
                                Pwm 5,3600,65535
                                Pwm 4,3500,65535
                                Do While Count(1)<5
                                Loop
                        Else
                                'Guessed Correction got better, Turn Left
                                Pwm 4,3600,65535
                                Pwm 5,3500,65535
                                Do While Count(0)<5
                                Loop
                        End If
                Else
                        'The error in the distance is worse than the error in the angle, Go Straight
                        Pwm 4,3600,65535
                        Pwm 5,3600,65535
                        Do While (Count(0)+Count(1))<10
                        Loop
                End If
                navGetPosition
```

```
            distFromGoal=Sqr((nextX-POSITION(0))^2 + (nextY-POSITION(1))^2)
            'Debug Float distFromGoal, " ", Float POSITION(0)," ", Float POSITION(1),"
",Float POSITION(2),Cr
        Loop

        LastX=nextX
        LastY=nextY
End Sub

        Sub TRACKING()
            High 34 'center distance sensor enable
            High 36 'blue light sensor enable
            High 18 'PAN motor enable
            Low 19 'PWM channel config
            Dim trk As Byte 'tracking loop counter
            Dim lt As Integer 'left blue light ADC value
            Dim ct As Integer 'right blue light ADC value
            Dim rt As Integer       'center blue light ADC value
            Dim dist As Integer  'distance from robot center target
            Dim headangle As Single 'head angle wrt body in radians  (left = pi/2, center =0,
right =-pi/2)
            Dim confidence As Integer 'percentage confidence
            For trk=0 To 4 'trys to resolve a fix on the intruder in 10 trys if not it boots out of
the sub
                lt = Tadin(4) 'sets lt as ADC value of left blue light sensor
                ct = Tadin(3) 'sets ct as ADC value of center blue light sensor
                rt = Tadin(2) 'sets rt as ADC value of right blue light sensor
                If (1.0*ct)<(lt+rt) Then 'changes PWM value if the left or right sensor has
value higher than center
                    If lt>rt Then SPWM=SPWM-0.1*(lt-ct) 'If lt > rt, head moves to
left percentage of difference between lt and ct
                    If rt>lt Then SPWM=SPWM+0.1*(rt-ct)      'If lt > rt, head moves
to left percentage of difference between lt and ct
                Endif
                If SPWM<1900 Then SPWM=1900 'left motion max limit
                If SPWM>5200 Then SPWM=5200 'right right max limit
                Pwm 3,SPWM,65535 'sets motor to new position
                dist = Longrange(Tadin(6)) 'coverts Longrange distance sensor ADC to
inches
                headangle = 3.379406-(0.0009519*SPWM) 'converts scan head PWM to
radians (left = pi/2, center =0, right =-pi/2)
                INTRUDER(2) = 100.0*(ct/400.0) 'calculates confidence of blue light
tracking based on center intensity and distance
                If INTRUDER(2) > 100 Then INTRUDER(2) = 100 'limits confidence to
100%
```

```
                If INTRUDER(2) < 20 Then SCAN 'requires a confidence to higher then
30% or it does a broad scan
                If INTRUDER(2) > 50 Then 'requires confidence to be higher then 50% to
report intruder position

        INTRUDER(0)=POSITION(0)+dist*Cos(headangle+POSITION(2)) 'calcs x cordinate of
intruder

        INTRUDER(1)=POSITION(1)+dist*Sin(headangle+POSITION(2)) 'calcs y cordinate of
intruder
                        Goto endtrack 'pointer to get out of FOR loop
                Else
                        INTRUDER(0)=0       'sets to zero if confidence is too low
                        INTRUDER(1)=0       'sets to zero if confidence is too low
                        INTRUDER(2)=0       'sets to zero if confidence is too low

                Endif
            Next
            endtrack:
        End Sub

        Sub SCAN()
            High 34 'center distance sensor enable
            High 36 'blue light sensor enable
            High 18 'PAN motor enable
            Low 19 'PWM channel config
            Dim i As Integer 'counter value
            Dim level As Integer 'threshold to continue scanning
                level = 40 'minimum level needed to try tracking
                For i=SPWM To 1900 Step -10         'moves the head all the way to the
left
                        If Tadin(3)> level Then Goto break   'breaks from the loop if the
level is met
                        Pwm 3,i,65535          'moves motor
                Next
                For i=1900 To 5200 Step 10  'scans from left to right
                        If Tadin(3)> level Then Goto break   'breaks from the loop if the
level is met
                        Pwm 3,i,65535
                Next
            For i=5200 To SPWM Step -10         'scans from Right To Left
                        If Tadin(3)> level Then Goto break   'breaks from the loop if the
level is met
                        Pwm 3,i,65535 'moves motor
                Next
            break:  'break pointer
```

```
            SPWM=i          'sets global SPWM to last position scanned
        End Sub

Function Longrange (C As Integer) As Integer 'converts ADC value of a longrange distance sensor to inches

        Longrange= 5197.9*(C)^-1.0003

End Function


Sub initRadio()
'This sub reads the unique radio identification number from EEPROM, opens communication with the radio attached to its serial port
'and clears the buffer
        SSN=Eeread(0,1)
        Opencom 1,115200,3,105,105
        Bclr 1,2
End Sub

Sub updateRawVoltages()
'This sub updates a global array containing the raw voltages for all analog channels
        Dim i As Integer
        For i=0 To 7
                RAW(i) = Tadin(i)
        Next
End Sub

Sub enableSensors()
'This sub enables distance sensors and blue light sensors
        High 33
        High 34
        High 35
        High 36
End Sub

Sub receiveRadioComm()
'This sub takes a message received and verified by the radio and parses it into various parts.
'The header contains the address, a function category, the function number within that category, and a data length
'followed by the amount of data claimed in the header. Once parsed into global and local variables, the function passes
'the message onto a delegation function which performs the operation requested.
        Dim funcID As Integer
        Dim dataLength As Integer
        Dim k As Integer
```

```
	If (Blen(1,0)>=7) Then 'When the length of the input is at least 7
		Geta 1,header,7 'Read the header
	End If

		For k=0 To 6
			header(k)=header(k)-48		'Take ASCII character and convert to
numerical value
		Next

	funcID=header(3)*10+header(4)				'Determine the function ID to
be executed
	dataLength=(header(5)*10+header(6))	'Determine how long the input string to the
function will be

	Do While (Blen(1,0)<dataLength)		'Wait until the all input is read
	Loop

	Geta 1, dataInput, dataLength			'Puts input values into an array
	Bclr 1,2 'Clear
buffer

	For k=0 To dataLength-1
		dataInput(k)=dataInput(k)-48 'Take ASCII character and convert to numerical
value
	Next
		commDelegate header(2),funcID
End Sub

Sub ackMsg()
		High 47
		Do While In(48)=1
		Loop
		Low 47
End Sub

Sub commDelegate(categoryID As Integer, functionID As Integer)
'This sub pushes the function number to the delegator of the requested category.
	Select  Case categoryID
			Case 0
				commCatStatus functionID
			Case 1
				commCatRole functionID
			Case 2
				commCatObstacle functionID
			Case 3
```

```
                commCatAutonomous functionID
        End Select
End Sub

        Sub commCatStatus(functionID As Integer)
        'This is the STATUS delegation function, it selects a function to execute based on the
function ID
                Select  Case functionID
                        Case 0
                                statusHandshake
                        Case 1
                                statusSendRaw
                        Case 2
                                statusSendPWM
                        Case 3
                                statusSendNav
                        Case 4
                                statusSendObstacle
                        Case 5
                                statusSendIntruder
                End Select
        End Sub

                Sub statusHandshake()
                'This sub responds to the server with a HELO message and the battery voltage to
confirm adequate life
                        Putstr 1,"5000011|HELO|"
                        If RAW(0)<999 Then Putstr 1,"0"
                        If RAW(0)<99 Then Putstr 1,"0"
                        If RAW(0)<9 Then Putstr 1,"0"
                        Putstr 1,Dec RAW(0),"|",Cr
                End Sub

                Sub statusSendRaw()
                'This sub responds to the server with the raw voltages for all analog channels
                        Dim i As Integer
                        Putstr 1,"5000142|"
                        For i=0 To 7
                                If RAW(i)<999 Then Putstr 1,"0"
                                If RAW(i)<99 Then Putstr 1,"0"
                                If RAW(i)<9 Then Putstr 1,"0"
                                Putstr 1,Dec RAW(i),"|"
                        Next
                        Putstr 1,Cr
                End Sub
```

```
		Sub statusSendPWM()
		'This sub responds to the server with the current PWM values for all PWM
channels
			Dim i As Integer
			Putstr 1,"5000226|"
			For i=0 To 3
				If PWMS(i)<9999 Then Putstr 1,"0"
				If PWMS(i)<999 Then Putstr 1,"0"
				If PWMS(i)<99 Then Putstr 1,"0"
				If PWMS(i)<9 Then Putstr 1,"0"
				Putstr 1,Dec PWMS(i),"|"
			Next
			Putstr 1,Cr
		End Sub

		Sub statusSendNav()
		'This sub responds to the server with the current position information of the robot
			Dim datLen As Byte
			datLen=Len(Dec POSITION(0)) + Len(Dec POSITION(1)) + Len(Dec
POSITION(2)) + 5 'Calculates the length of all position data and markup
			Putstr 1,"50003",Dec (datLen),"|",Dec POSITION(0),"|",Dec
POSITION(1),"|",Dec POSITION(2),"|",Cr 'Returns Position info
		End Sub

		Sub statusSendObstacle()
		'This sub responds to the server with all information pertaining to obstacle
avoidance
		End Sub

		Sub statusSendIntruder()
			Dim datLen As Byte
			datLen=Len(Dec INTRUDER(0)) + Len(Dec INTRUDER(1)) + Len(Dec
INTRUDER(2)) + 5 'Calculates the length of all position data and markup
			Putstr 1,"50005",Dec (datLen),"|",Dec INTRUDER(0),"|",Dec
INTRUDER(1),"|",Dec INTRUDER(2),"|",Cr 'Returns Intruder Position info
		End Sub

	Sub commCatRole(functionID As Integer)
	'This is the ROLE delegation function, it executes the function requested by the function
ID
		Select  Case functionID
			Case 0
				roleSleep
			Case 1
				roleWait
			Case 2
```

```
                    roleActive
            End Select
        End Sub

            Sub roleSleep()
            'This sub puts the robot in SLEEP mode
            End Sub

            Sub roleWait()
            'This sub puts the robot in WAIT mode
            End Sub

            Sub roleActive()
            'This sub puts the robot in ACTIVE mode
            End Sub

        Sub commCatObstacle(functionID As Integer)
        'This is the OBSTACLE delegation function, it executes the function requested by the
function ID
            'Select  Case functionID
            '       Case 0
            '       Case 1
            '       Case 2
            '       Case 3
            '       Case 4
            'End Select
        End Sub

        Sub commCatAutonomous(functionID As Integer)
        'This is the AUTONOMOUS delegation function, it executes the function requested by
the function ID
            Select  Case functionID
                    Case 0
                        autonomousMatrix
                    Case 1
                        TurnToHeading
                    Case 2
                        Backup
            '       Case 3
            '       Case 4
            End Select
        End Sub

        Sub autonomousMatrix()
            Dim i As Integer
            Dim j As Integer
```

```
        i=0
        j=0
        Do While i<54
            PositionMatrix(j,0)=dataInput(i)*100+dataInput(i+1)*10+dataInput(i+2)

    PositionMatrix(j,1)=dataInput(i+3)*100+dataInput(i+4)*10+dataInput(i+5)
            j=j+1
            i=i+6
        Loop
        delegatePosMatrix
    End Sub

    Sub TurnToHeading ()
        Dim Heading As Single
        Heading =
dataInput(0)+dataInput(1)/10+dataInput(2)/100+dataInput(3)/1000+dataInput(4)/10000+dataInp
ut(5)/100000
        If POSITION(2)>6.28318530 Then POSITION(2) = POSITION(2) MOD
6.28318530
         If POSITION(2)>Heading Then
                Pwm 4,3450,65535
                Pwm 5,3550,65535
                    Do While
(Count(0)+Count(1))<(4.304*Fabs(POSITION(2)-Heading)/0.022776)
                    Loop
                Pwm 4,3500,65535
                Pwm 5,3500,65535
                Delay 100
                POSITION(2)=(Heading-
(((0.022776/4.304)*(Count(0)+Count(1)))-POSITION(2)))
            Else
                Pwm 4,3550,65535
                Pwm 5,3450,65535
                    Do While Count(0)+Count(1)<(4.304*Fabs(POSITION(2)-
Heading)/0.022776)
                    Loop
                Pwm 4,3500,65535
                Pwm 5,3500,65535
                Delay 100

    POSITION(2)=(Heading+(((0.022776/4.304)*(Count(0)+Count(1)))-POSITION(2)))
        Endif
        Countreset 0
        Countreset 1
    End Sub
```

```
	Sub Backup ()
		Dim Inches As Integer
		Inches = dataInput(0)*100+dataInput(1)*10+dataInput(2)
		Pwm 4,3450,65535
		Pwm 5,3450,65535
			Do While ((Count(0)+Count(1))/2)<(Inches/0.022776)
			Loop
		Pwm 4,3500,65535
		Pwm 5,3500,65535
		Delay 100
		POSITION(0) = POSITION(0) -
((Count(0)+Count(1))/2*0.022776)*(Cos(POSITION(2)))
		POSITION(1) = POSITION(1) -
((Count(0)+Count(1))/2*0.022776)*(Sin(POSITION(2)))
		Countreset 0
		Countreset 1
	End Sub
```

Appendix F: Radio Cubloc Code

```
Const Device =cb220
Dim SSN As Integer
SSN = Eeread(0,1)

        ' DATA RECEIVE SPECIFIC
Dim HEADR(7) As Byte
Dim DAT(104) As Byte
Dim ROBOID As Integer
Dim HEADR_CODE As Integer
Dim DAT_LEN As Integer
Dim REC_INDEX As Integer
Input 13
Output 12
Low 12

Opencom 1,115200,3,105,105
Opencom 0,115200,3,105,105
Bclr 1,2
Bclr 0,2

On Recv1 Gosub RECEIVE
Set Until 1,7


Set Debug Off
Debug "RADIO ID: ",Dec SSN,Cr
Low 12


Do
        If In(13)=1 Then
                Low 12
        Endif
Loop

RECEIVE:
        If (Blen(1,0)>=7) Then
                Geta 1,HEADR,7

                For REC_INDEX=0 To 6
                        HEADR(REC_INDEX)=HEADR(REC_INDEX)-48
                        Debug Dec HEADR(REC_INDEX)," "

                Next
```

```
                ROBOID = HEADR(0)*10+HEADR(1)

                HEADR_CODE=HEADR(3)*10+HEADR(4)

                DAT_LEN=(HEADR(5)*10+HEADR(6))

                Do While (Blen(1,0)<DAT_LEN)
                Loop
                Geta 1, DAT, DAT_LEN
                Bclr 1,2

                For REC_INDEX=0 To DAT_LEN-1
                        DAT(REC_INDEX)=DAT(REC_INDEX)-48
                Next

                If ROBOID=SSN Then

                        High 12

                        Debug Cr,"CONTROL HEADER",Cr

                        For REC_INDEX=0 To 6
                                Debug Dec HEADR(REC_INDEX)," "
                                Putstr 0,HEADR(REC_INDEX)+48

                        Next

                        Debug Cr,"DATA",Cr

                        For REC_INDEX=0 To DAT_LEN-1
                                Debug Dec DAT(REC_INDEX)," "
                                Putstr 0,DAT(REC_INDEX)+48
                        Next

                        Debug Cr

                Else
                        Return

                End If

        End If

        Bclr 1,2
Return
```

Appendix G: Base Station Code

PhysicalRobot

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;

namespace RoboSim
{
    class PhysicalRobot : IDisposable
    {
        //Robot Information
        private byte robotID; //ID Number Unique to Each Robot
        private byte robotStatus; //0=Active 1=Paused 2=Sleeping
        private string robotName; //Name Used to Describe Robot (Not Unique)
        private bool inSimulation; //0=No  1=yes

        //Communication
        private short commAddress; //16-bit Hex Address Used for Radios
        private string commStatus; //0=Not Initialized 1=Failed
2=Initializing 3=Successful
        private byte commStatColor; //Represents the text color status will
be displayed in

        //Raw Analog Data
        private short battVoltage=0; //0-1024 Battery Voltage
        private short leftLightVoltage=0; //0-1024 Left Blue Light Sensor
Voltage
        private short centerLightVoltage=0; //0-1024 Center Blue Light Sensor
Voltage
        private short rightLightVoltage=0; //0-1024 Right Blue Light Sensor
Voltage
        private short leftDistanceVoltage=0; //0-1024 Left Distance Sensor
Voltage
        private short centerDistanceVoltage=0; //0-1024 Center Distance
Sensor Voltage
        private short rightDistanceVoltage=0; //0-1024 Right Distance Sensor
Voltage
        private short auxVoltage=0; //0-1024 Auxilary Port Voltage

        //Raw PWM Data
        private int sensorPanningMotor=0; //Sensor Panning Motor PWM
        private int leftDriveMotor=0; //Left Driving Motor PWM
        private int rightDriveMotor=0; //Right Driving Motor PWM
        private int battCharge=0; //Is Battery Charging?

        //Navigation
        private int posX=0; //X Position on Environment
        private int posY=0; //Y Position on Environment
        private double heading=0.0; //What is the Robots Heading?

        private int startX=-1; //Where should I start X?
        private int startY=-1;  //Where should I start Y?
```

```
private double startHeading = 90; //Where should I Start facing?

private int[,] PositionMatrix;


//Obstacle Detection
private int intruderX = 0;
private int intruderY = 0;
private int confidenceInterval = 0;


//Functions / Methods

//Constructors
public PhysicalRobot()
{
    robotID = 0;
    robotStatus = 3;
}
public PhysicalRobot(byte newRobotID)
{
    robotID = newRobotID;
    setCommStatus(0);
    battVoltage = 0;
}

//Destructors
public void Dispose()
{
}

//Get Properties
public byte RobotID
{
    get
    {
        return robotID;
    }
}
public string RobotName
{
    get
    {
        return robotName;
    }
}
public short RobotCommAddress
{
    get
    {
        return commAddress;
    }
}

//Accessors
public short RobotCommAddress2()
{
```

```
    return commAddress;
}
public string getCommStatus()
{
    return commStatus;
}
public byte getCommStatColor()
{
    return commStatColor;
}
public byte getStatus()
{
    return robotStatus;
}
public byte getRobotID()
{
    return robotID;
}
public override string ToString()
{
    return robotName;
}
public bool isInSimulation()
{
    return inSimulation;
}

public double getBattVoltage()
{
    return (double)battVoltage*10/1023;
}
public double getLeftLightVoltage()
{
    return (double)leftLightVoltage*5/1023;
}
public double getCenterLightVoltage()
{
    return (double)centerLightVoltage*5/1023;
}
public double getRightLightVoltage()
{
    return (double)rightLightVoltage*5/1023;
}
public double getLeftDistanceVoltage()
{
    return (double)leftDistanceVoltage*5/1023;
}
public double getCenterDistanceVoltage()
{
    return (double)centerDistanceVoltage*5/1023;
}
public double getRightDistanceVoltage()
{
    return (double)rightDistanceVoltage*5/1023;
}
public double getAuxVoltage()
{
```

```
    return (double)auxVoltage*5/1023;
}

public int getSensorPanningMotorPWM()
{
    return sensorPanningMotor;
}
public int getLeftDriveMotorPWM()
{
    return leftDriveMotor;
}
public int getRightDriveMotorPWM()
{
    return rightDriveMotor;
}
public int getBattChargePWM()
{
    return battCharge;
}

public int getPosX()
{
    return posX;
}
public int getPosY()
{
    return posY;
}
public double getHeading()
{
    return heading;
}
public int[,] getPositionMatrix()
{
    return PositionMatrix;
}

public int getIntruderX()
{
    return intruderX;
}
public int getIntruderY()
{
    return intruderY;
}
public int getConfidenceInterval()
{
    return confidenceInterval;
}

public int getStartX()
{
    return startX;
}
public int getStartY()
{
    return startY;
```

```
        }
        public double getStartHeading()
        {
            return startHeading;
        }


        //Modifiers
        public void setName(string newName)
        {
            robotName = newName;
        }
        public void setInSimulation(bool newValue)
        {
            inSimulation = newValue;
        }
        public void setCommAddress(short newCommAdd)
        {
            commAddress = newCommAdd;
        }
        public void setCommStatus(short newStatus)
        {
            switch (newStatus)
            {
                case 0: { commStatus = "Not Initialized"; commStatColor = 0;
break; }
                case 1: { commStatus = "Failed"; commStatColor = 1; break; }
                case 2: { commStatus = "Initializing..."; commStatColor = 2;
break; }
                case 3: { commStatus = "Connected"; commStatColor = 3; break;
}
            }
        }
        public void setStatus(byte newStatus)
        {
            robotStatus = newStatus;
        }
        public void setAnalogs(ref string[] myValues)
        {
            battVoltage = short.Parse(myValues[1]);
            rightLightVoltage = short.Parse(myValues[2]);
            centerLightVoltage = short.Parse(myValues[3]);
            leftLightVoltage = short.Parse(myValues[4]);
            rightDistanceVoltage = short.Parse(myValues[5]);
            centerDistanceVoltage = short.Parse(myValues[6]);
            leftDistanceVoltage = short.Parse(myValues[7]);
            auxVoltage = short.Parse(myValues[8]);
        }
        public void setPWMS(ref string[] myValues)
        {
            leftDriveMotor = int.Parse(myValues[1]);
            rightDriveMotor = int.Parse(myValues[2]);
            sensorPanningMotor = int.Parse(myValues[3]);
            battCharge = int.Parse(myValues[4]);
        }
        public void setPosition(ref string[] myValues)
        {
```

```
            posX = (int)(double.Parse(myValues[1]));
            posY = (int)(double.Parse(myValues[2]));
            heading = double.Parse(myValues[3]);
        }
        public void setStartPos(int newX, int newY)
        {
            startX = newX;
            startY = newY;
        }
        public void setStartHeading(int newValue)
        {
            startHeading = newValue;
        }
        public void setPositionMatrix(ref int[,] newMatrix)
        {
            PositionMatrix = newMatrix;
        }
        public void setIntruderPosition(ref string[] myValues)
        {
            intruderX=(int)(double.Parse(myValues[1]));
            intruderY=(int)(double.Parse(myValues[2]));
            confidenceInterval=(int)(double.Parse(myValues[3]));
        }

    }
}
```

Program

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace RoboSim
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new frmMainControl());
        }
    }
}
```

frmRobotHealth

```
using System;
```

```
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Threading;

namespace RoboSim
{
    public partial class frmRobotHealth : Form
    {
        ArrayList activeRobots;
        Object waste=null;
        EventArgs waste2=null;
        System.IO.Ports.SerialPort serialComm;

        //Constructor
        public frmRobotHealth(ref ArrayList myRobots, ref
System.IO.Ports.SerialPort newSerialComm)
        {
            InitializeComponent();
            serialComm = newSerialComm;
            activeRobots = myRobots;
            foreach (object robot in activeRobots)
            {
                lstRunningRobots.Items.Add(robot);
            }
            lstRunningRobots.SelectedIndex = 0;
            lstRunningRobots.Refresh();
            tmrUpdate.Interval = 1000;
            tmrUpdate.Tick+=new EventHandler(updateData);
            startTimer();
        }

        //Destructor
        public void Dispose()
        {
            tmrUpdate.Dispose();
        }
        public void startTimer()
        {
            tmrUpdate.Start();
        }

        //Functions-Methods
        private void updateData(object o, EventArgs eargs)
        {
            PhysicalRobot selRobot =
(PhysicalRobot)activeRobots[lstRunningRobots.SelectedIndex];
            lblRobotID.Text = selRobot.getRobotID().ToString();
            lblRobotName.Text = selRobot.ToString();
            switch (selRobot.getStatus())
            {
                case 0: { lblScenarioRole.Text = "Active"; break; }
                case 1: { lblScenarioRole.Text = "Paused"; break; }
                case 2: { lblScenarioRole.Text = "Sleeping"; break; }
```

```
            }
            lblCommAddress.Text = selRobot.RobotCommAddress2().ToString();
            lblCommStatus.Text = selRobot.getCommStatus();
            lblBattVoltage.Text = selRobot.getBattVoltage().ToString("n"+2);
            lblLeftLightVoltage.Text =
selRobot.getLeftLightVoltage().ToString("n"+2);
            lblCenterLightVoltage.Text =
selRobot.getCenterLightVoltage().ToString("n"+2);
            lblRightLightVoltage.Text =
selRobot.getRightLightVoltage().ToString("n"+2);
            lblRightDistanceVoltage.Text =
selRobot.getRightDistanceVoltage().ToString("n"+2);
            lblCenterDistanceVoltage.Text =
selRobot.getCenterDistanceVoltage().ToString("n"+2);
            lblLeftDistanceVoltage.Text =
selRobot.getLeftDistanceVoltage().ToString("n"+2);
            lblAuxVoltage.Text = selRobot.getAuxVoltage().ToString("n"+2);
            lblLeftDriveMotor.Text =
selRobot.getLeftDriveMotorPWM().ToString();
            lblRightDriveMotor.Text =
selRobot.getRightDriveMotorPWM().ToString();
            lblSensorPanningMotor.Text =
selRobot.getSensorPanningMotorPWM().ToString();
            lblBattCharge.Text = selRobot.getBattChargePWM().ToString();
            lblXPos.Text = selRobot.getPosX().ToString();
            lblYPos.Text = selRobot.getPosY().ToString();
            lblH.Text = selRobot.getHeading().ToString();
        }

        //Events
        private void lstRunningRobots_SelectedIndexChanged(object sender,
EventArgs e)
        {
            updateData(waste,waste2);
        }
        private void btnClose_Click(object sender, EventArgs e)
        {
            this.Visible = false;
            tmrUpdate.Stop();
        }
    }
}
```

frmRobotFileNotExist

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace RoboSim
{
    public partial class frmRobotFileNotExist : Form
```

```
    {
        public frmRobotFileNotExist()
        {
            InitializeComponent();
        }
    }
}
```

frmConfigureRobots

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace RoboSim
{
    public partial class frmConfigureRobots : Form
    {
        private ArrayList mine;
        public frmConfigureRobots()
        {
            InitializeComponent();
        }

        public void setRoboList(ref ArrayList newList)
        {
            mine = newList;
            dgvRobotDisplay.DataSource = newList;
            dgvRobotDisplay.Refresh();
        }
    }
}
```

frmAddNewRobot

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace RoboSim
{
    public partial class frmAddNewRobot : Form
    {
        public frmAddNewRobot()
        {
            InitializeComponent();
        }
    }
```

```
}
```

CommTest

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;

namespace RoboSim
{
    public partial class CommTest : Form
    {
        SerialPort serialComm;
        public CommTest(ref SerialPort newSerialComm)
        {
            InitializeComponent();
            serialComm = newSerialComm;

        }

        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                serialComm.ReadTimeout = 1000;
                serialComm.WriteTimeout = 1000;
                serialComm.Write(textBox1.Text);
                string x = serialComm.ReadLine();
                label2.Text = x;
            }
            catch
            {
                label2.Text = "Timeout";
            }
        }
    }
}
```

frmNewScenario

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.IO.Ports;
```

```
namespace RoboSim
{
    public partial class frmNewScenario : Form
    {
        ArrayList loadedRobots;
        double myHeight;
        double myWidth;
        int pixelsPerUnit;
        double pixelsPerUnitD;

        int tempX=-1;
        int tempY=-1;
        int tempHeading = 180;

        //Constructors
        public frmNewScenario(ref ArrayList myRobots, ref SerialPort
newSerialComm)
        {
            InitializeComponent();
            loadedRobots = myRobots;
            serialComm = newSerialComm;
            txtLength.Text = "0";
            txtWidth.Text = "0";
            foreach (PhysicalRobot robot in loadedRobots)
            {
                clistBoxRobots.Items.Add(robot);
                lstPlaceRobots.Items.Add(robot);
            }
            clistBoxRobots.SetSelected(0, true);
            lstPlaceRobots.SetSelected(0, true);
        }

        //Functions-Methods
        private void updateCommStatus(int index)
        {
            lblCommStat.Text =
((PhysicalRobot)loadedRobots[index]).getCommStatus();
            byte color =
((PhysicalRobot)loadedRobots[clistBoxRobots.SelectedIndex]).getCommStatColor(
);
            switch (color)
            {
                case 0: { lblCommStat.ForeColor =
System.Drawing.Color.Yellow; break; }
                case 1: { lblCommStat.ForeColor = System.Drawing.Color.Red;
break; }
                case 2: { lblCommStat.ForeColor =
System.Drawing.Color.Yellow; break; }
                case 3: { lblCommStat.ForeColor = System.Drawing.Color.Green;
break; }
            }
        }
        private void updateTeamStatus(object robot)
        {
            PhysicalRobot myRobot = (PhysicalRobot)robot;
            lstActive.Items.Remove(myRobot);
            lstPaused.Items.Remove(myRobot);
```

```
            lstSleep.Items.Remove(myRobot);
            switch (myRobot.getStatus())
            {
                case 0: { lstActive.Items.Add(myRobot); break; }
                case 1: { lstPaused.Items.Add(myRobot); break; }
                case 2: { lstSleep.Items.Add(myRobot); break; }
            }
        }
        private void removeRobotFromTeam(object robot)
        {
            PhysicalRobot myRobot = (PhysicalRobot)robot;
            lstActive.Items.Remove(myRobot);
            lstPaused.Items.Remove(myRobot);
            lstSleep.Items.Remove(myRobot);
        }
        private bool handshake(object newHex)
        {
            try
            {
                string newDestHex = newHex.ToString();
                serialComm.Write(newDestHex+"00000");
                serialComm.WriteTimeout = 1000;
                serialComm.ReadTimeout = 1000;
                string response = serialComm.ReadLine();
                string[] respParsed = response.Split('|');
                response = response.Substring(8, response.Length - 8);
                if (respParsed[1].StartsWith("HELO"))
                    return true;
                else
                    return false;
            }
            catch (Exception ex)
            {
                string temp = ex.StackTrace;
                return false;
            }
        }
        private void fail()
        {

((PhysicalRobot)loadedRobots[clistBoxRobots.SelectedIndex]).setCommStatus(1);
            removeRobotFromTeam(loadedRobots[clistBoxRobots.SelectedIndex]);
            clistBoxRobots.SetItemChecked(clistBoxRobots.SelectedIndex,
false);
            string
name=((PhysicalRobot)loadedRobots[clistBoxRobots.SelectedIndex]).ToString();
            MessageBox.Show("Communication tests for " + name + " have
failed");
        }
        public double feetToInches(double feet)
        {
            return feet * 12;
        }
        public double metersToCm(double meters)
        {
            return meters * 100;
        }
```

```
        private bool canRobotBePlaced(int xCoord, int yCoord)
        {
                bool canIDoThis = true;
                if (xCoord >= 15 && yCoord >= 15 && yCoord <= pnlEnv.Height -
(15) && xCoord <= pnlEnv.Width - (15))
                {
                    return canIDoThis;
                }
                else
                    return false;
        }
        private bool canRobotBePlaced2(int xCoord, int yCoord)
        {
            bool canIDoThis = true;
            try
            {
                foreach (PhysicalRobot test in lstPlaceRobots.Items)
                {
                    if
(!test.Equals(lstPlaceRobots.Items[lstPlaceRobots.SelectedIndex]))
                    {
                        if (Math.Abs(test.getStartX() - xCoord) < 20 *
pixelsPerUnit && Math.Abs(test.getStartY() - yCoord) < 20 * pixelsPerUnit)
                            canIDoThis = false;
                    }
                }
                return canIDoThis;
            }
            catch
            {
                return false;
            }
        }
        public int getEnvWidth()
        {
            return int.Parse(txtWidth.Text);
        }
        public int getEnvLength()
        {
            return int.Parse(txtLength.Text);
        }

        //Events
        private void clistBoxRobots_SelectedIndexChanged(object sender,
EventArgs e)
        {
            int index = clistBoxRobots.SelectedIndex;
            PhysicalRobot currRobot = (PhysicalRobot)loadedRobots[index];
            updateCommStatus(clistBoxRobots.SelectedIndex);
            lblRobotID.Text = currRobot.getRobotID().ToString();
            lblRobotName.Text = currRobot.ToString();
            lblRobotCommNum.Text = currRobot.RobotCommAddress2().ToString();
            if (currRobot.getCommStatus().StartsWith("Failed"))
                clistBoxRobots.SetItemChecked(clistBoxRobots.SelectedIndex,
false);
            if (clistBoxRobots.GetItemChecked(clistBoxRobots.SelectedIndex))
            {
```

```
                lstRobotStatus.Enabled=true;
                lstRobotStatus.SelectedIndex = currRobot.getStatus();
            }
            else
            {
                lstRobotStatus.Enabled=false;
            }
        }
        private void clistBoxRobots_ItemCheck(object sender,
ItemCheckEventArgs e)
        {
            if (clistBoxRobots.GetItemChecked(clistBoxRobots.SelectedIndex))
            {
                lstRobotStatus.Enabled = false;

removeRobotFromTeam(loadedRobots[clistBoxRobots.SelectedIndex]);

((PhysicalRobot)loadedRobots[clistBoxRobots.SelectedIndex]).setInSimulation(f
alse);
            }
            else
            {
                string
newHex=((PhysicalRobot)loadedRobots[clistBoxRobots.SelectedIndex]).RobotCommA
ddress2().ToString();

((PhysicalRobot)loadedRobots[clistBoxRobots.SelectedIndex]).setCommStatus(2);
                updateCommStatus(clistBoxRobots.SelectedIndex);
                lstRobotStatus.Enabled = true;
                lstRobotStatus.SelectedIndex =
((PhysicalRobot)loadedRobots[clistBoxRobots.SelectedIndex]).getStatus();
                updateTeamStatus(loadedRobots[clistBoxRobots.SelectedIndex]);
                if (handshake(newHex))
                {

((PhysicalRobot)loadedRobots[clistBoxRobots.SelectedIndex]).setCommStatus(3);

((PhysicalRobot)loadedRobots[clistBoxRobots.SelectedIndex]).setInSimulation(t
rue);
                }
                else
                {
                    fail();
                }
                updateCommStatus(clistBoxRobots.SelectedIndex);
            }
        }
        private void lstRobotStatus_SelectedIndexChanged(object sender,
EventArgs e)
        {

((PhysicalRobot)loadedRobots[clistBoxRobots.SelectedIndex]).setStatus(byte.Pa
rse(lstRobotStatus.SelectedIndex.ToString()));
            updateTeamStatus(loadedRobots[clistBoxRobots.SelectedIndex]);
        }
        private void btnCancel_Click(object sender, EventArgs e)
        {
```

```
        }
        private void btnNewScenario_Click(object sender, EventArgs e)
        {
            try
            {
                int NumRobots;
                int MatrixDistance;
                int RobotDistance;
                int[,] tempMatrix = new int[10, 2];
                double Heading;
                NumRobots = lstPlaceRobots.Items.Count;
                PhysicalRobot tempRobot;
                string id;
                for (int i = 0; i < NumRobots; i++)
                {
                    tempRobot = (PhysicalRobot)(lstPlaceRobots.Items[i]);
                    id = tempRobot.getRobotID().ToString();
                    serialComm.Write(id + "00300");
                    string response = serialComm.ReadLine();
                    string[] output = response.Split('|');
                    tempRobot.setPosition(ref output);
                    tempMatrix[0, 0] = tempRobot.getPosX();
                    tempMatrix[0, 1] = tempRobot.getPosY();
                    tempMatrix[1, 0] = tempRobot.getPosX();
                    tempMatrix[1, 1] = tempRobot.getPosY() + 010;
                    for (int j = 2; j < 10; j++)
                    {
                        tempMatrix[j, 0] = (int)(tempRobot.getStartX() * (j /
7.0));
                        tempMatrix[j, 1] = (int)(tempRobot.getStartY() * (j /
7.0));
                    }
                    tempRobot.setPositionMatrix(ref tempMatrix);
                    string PosMatrix;
                    PosMatrix = tempMatrix[0, 0].ToString() + "|";
                    PosMatrix += tempMatrix[0, 1].ToString() + "|";
                    for (int n = 1; n < 10; n++)
                    {
                        PosMatrix += tempMatrix[i, 0].ToString() + "|";
                        PosMatrix += tempMatrix[i, 1].ToString() + "|";
                    }
                    serialComm.Write(id + "300" + PosMatrix.Length.ToString()
+ PosMatrix);
                    string ack = serialComm.ReadLine();
                    MatrixDistance = (int)(Math.Sqrt((((tempMatrix[3, 0] -
tempMatrix[0, 0]) ^ 2) + ((tempMatrix[3, 1] - tempMatrix[0, 1]) ^ 2))));
                    for (int m = 1; m < NumRobots; m++)
                    {
                        RobotDistance = (int)Math.Sqrt((((tempRobot.getPosX()
- tempMatrix[0, 0]) ^ 2 + (tempRobot.getPosY() - tempMatrix[0, 1]) ^ 2)));
                        while (RobotDistance < MatrixDistance)
                        {
                            RobotDistance =
(int)Math.Sqrt((((tempRobot.getPosX() - tempMatrix[0, 0]) ^ 2 +
(tempRobot.getPosY() - tempMatrix[0, 1]) ^ 2)));
                            for (int k = 0; k < 100000; k++)
                            {
```

```
                            }
                        }
                    }
                }

                tempRobot = (PhysicalRobot)(lstPlaceRobots.Items[NumRobots -
1]);
                while (tempRobot.getPosX() + 1.5 < tempMatrix[9, 0] &&
tempRobot.getPosY() + 1.5 < tempMatrix[9, 1])
                {
                    tempRobot.getPosX();
                    tempRobot.getPosY();
                    for (int k = 0; k < 100000; k++)
                    {
                    }
                }
                for (int k = 0; k < 100000; k++)
                {
                }
                for (int i = 0; i < NumRobots; i++)
                {
                    tempRobot = (PhysicalRobot)(lstPlaceRobots.Items[i]);
                    Heading = tempRobot.getStartHeading() * (Math.PI / 180);
                    id = tempRobot.getRobotID().ToString();
                    serialComm.Write(id + "30106" +
Heading.ToString().Substring(0, 6));
                    string ack = serialComm.ReadLine();
                }
            }
            catch
            {
            }
        }

        private void pnlEnv_MouseClick(object sender, MouseEventArgs e)
        {
            tempX = -1;
            tempY = -1;
            try
            {
                if (canRobotBePlaced(e.X, e.Y))
                {
                    //if (canRobotBePlaced2(e.X,e.Y))
                    //{

((PhysicalRobot)lstPlaceRobots.Items[lstPlaceRobots.SelectedIndex]).setStartP
os((int)(e.X/pixelsPerUnitD), (int)(e.Y/pixelsPerUnitD));
                    tckbrHeading.Enabled = true;
                    tckbrHeading.Value =
(int)(((PhysicalRobot)lstPlaceRobots.Items[lstPlaceRobots.SelectedIndex]).get
StartHeading());
                    pnlEnv.Refresh();
                    //}
                    //else
                    //{
                    //    foreach (PhysicalRobot test in
lstPlaceRobots.Items)
```

```
                    //    {
                    //        if (Math.Abs(test.getStartX() - xCoord) < 20 *
pixelsPerUnit && Math.Abs(test.getStartY() - yCoord) < 20 * pixelsPerUnit)


                    //    }
                    //}
                }

            }
            catch
            {
                MessageBox.Show("Please select a robot. If there are no
robots in the list, consult the Robots tab");
            }
        }
        private void pnlEnv_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            GraphicsPath myG = new GraphicsPath();
            Matrix mat = new Matrix();
            if (tempX >= 0 && tempY >= 0)
            {
                mat.RotateAt(tempHeading, new PointF(tempX, tempY));
                myG.AddRectangle(new Rectangle((tempX - 2 * pixelsPerUnit),
(tempY - 4 * pixelsPerUnit), (4 * pixelsPerUnit), (6 * pixelsPerUnit)));
                myG.AddRectangle(new Rectangle((tempX - 3 * pixelsPerUnit),
(tempY - 5 * pixelsPerUnit), (1 * pixelsPerUnit), (9 * pixelsPerUnit)));
                myG.AddRectangle(new Rectangle((tempX + 2 * pixelsPerUnit),
(tempY - 5 * pixelsPerUnit), (1 * pixelsPerUnit), (9 * pixelsPerUnit)));
                myG.Transform(mat);
                if (canRobotBePlaced(tempX, tempY))
                    g.DrawPath(new Pen(Color.Blue), myG);
                else
                    g.DrawPath(new Pen(Color.Red), myG);
            }


            foreach (PhysicalRobot currRobot in lstPlaceRobots.Items)
            {
                int x=(int)(currRobot.getStartX()*pixelsPerUnitD);
                int y=(int)(currRobot.getStartY()*pixelsPerUnitD);

                if (x >= 0 && y >= 0)
                {
                    myG = new GraphicsPath();
                    mat = new Matrix();
                    mat.RotateAt((int)(currRobot.getStartHeading()),new
PointF(x,y));
                    myG.AddRectangle(new Rectangle((x - 2 * pixelsPerUnit),
(y - 4 * pixelsPerUnit), (4 * pixelsPerUnit), (6 * pixelsPerUnit)));
                    myG.AddRectangle(new Rectangle((x - 3 * pixelsPerUnit),
(y - 5 * pixelsPerUnit), (1 * pixelsPerUnit), (9 * pixelsPerUnit)));
                    myG.AddRectangle(new Rectangle((x + 2 * pixelsPerUnit),
(y - 5 * pixelsPerUnit), (1 * pixelsPerUnit), (9 * pixelsPerUnit)));
                    myG.Transform(mat);
```

```
                    if
(lstPlaceRobots.Items[lstPlaceRobots.SelectedIndex].Equals(currRobot))
                        g.DrawPath(new Pen(Color.Blue), myG);
                    else
                        g.DrawPath(new Pen(Color.Black), myG);

                }
            }
        }
        private void btnRemove_Click(object sender, EventArgs e)
        {

((PhysicalRobot)lstPlaceRobots.Items[lstPlaceRobots.SelectedIndex]).setStartP
os(-1, -1);

((PhysicalRobot)lstPlaceRobots.Items[lstPlaceRobots.SelectedIndex]).setStartH
eading(180);
            pnlEnv.Refresh();
        }
        private void lstPlaceRobots_SelectedIndexChanged(object sender,
EventArgs e)
        {
            PhysicalRobot loadValues =
((PhysicalRobot)lstPlaceRobots.Items[lstPlaceRobots.SelectedIndex]);
            if (loadValues.getStartY() < 0 || loadValues.getStartX() < 0)
                tckbrHeading.Enabled = false;
            else
                tckbrHeading.Enabled = true;
            tckbrHeading.Value = (int)(loadValues.getStartHeading());
            tempHeading = tckbrHeading.Value;
            pnlEnv.Refresh();
        }
        private void tckbrHeading_Scroll(object sender, EventArgs e)
        {

((PhysicalRobot)lstPlaceRobots.Items[lstPlaceRobots.SelectedIndex]).setStartH
eading(tckbrHeading.Value);
            tempHeading = tckbrHeading.Value;
            pnlEnv.Refresh();
        }
        private void pnlEnv_MouseMove(object sender, MouseEventArgs e)
        {
            tempX = e.X;
            tempY = e.Y;
            pnlEnv.Refresh();
        }
        private void pnlEnv_MouseLeave(object sender, EventArgs e)
        {
            Cursor.Show();
            tempX = -1;
            tempY = -1;
            pnlEnv.Refresh();
        }
        private void pnlEnv_MouseEnter(object sender, EventArgs e)
        {
            Cursor.Hide();
        }
```

```
        private void tabSwitcher_SelectedIndexChanged(object sender,
EventArgs e)
        {
            if (tabSwitcher.SelectedIndex == 3)
            {
                double envHeight = double.Parse(txtLength.Text);
                double envWidth = double.Parse(txtWidth.Text);
                if (envWidth>=envHeight)
                {
                    myHeight = envHeight;
                    myWidth = envWidth;
                }
                else
                {
                    myHeight = envWidth;
                    myWidth = envHeight;
                }
                if (0 == 0)
                {
                    myHeight = feetToInches(myHeight);
                    myWidth = feetToInches(myWidth);
                }
                else
                {
                    myHeight = metersToCm(myHeight);
                    myWidth = metersToCm(myWidth);
                }
                double ratio = myWidth / myHeight;
                if (ratio <= 2)
                {
                    pnlEnv.Height = 150;
                    pixelsPerUnitD = (150 / myHeight);
                    pnlEnv.Width = (int)(pixelsPerUnitD * myWidth);
                }
                else
                {
                    int tempHeight = 150;
                    pixelsPerUnitD = (150 / myHeight);
                    while (pixelsPerUnitD * myWidth > 300)
                    {
                        tempHeight--;
                        pixelsPerUnitD = (tempHeight / myHeight);
                    }
                    pnlEnv.Width =  (int)(pixelsPerUnitD * myWidth);
                    pnlEnv.Height = (int)(pixelsPerUnitD * myHeight);
                }
                pixelsPerUnit = (int)pixelsPerUnitD;
                if (pixelsPerUnit == 0)
                    pixelsPerUnit = 1;
            }
        }
    }
}
```

frmMainControl

```
using System;
using System.IO;
using System.IO.Ports;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.Drawing.Drawing2D;



namespace RoboSim
{
    public partial class frmMainControl : Form
    {
        private ArrayList loadedRobots;
        frmRobotHealth roboHealth;
        private double envWidth, envLength, pixelsPerUnitD;
        private int pixelsPerUnit;

        //Constructors
        public frmMainControl()
        {
            InitializeComponent();
            loadedRobots = loadRobots();
            initRadio();
            roboHealth = new frmRobotHealth(ref loadedRobots, ref
serialComm);
            roboHealth.Visible = false;
            tmrUpdate.Interval = 700;
            tmrUpdate.Tick += new EventHandler(updateData);
            serialComm.WriteTimeout = 1000;
            serialComm.ReadTimeout = 1000;
        }

        //Functions-Methods
        private ArrayList loadRobots()
        {
            string directory = @"C:\ERIC\ISP\RoboSim v.1.0\RoboSim
v.1.0\Robots\";
            string fileName = "index.ini";
            FileStream indexFile = new FileStream((directory + fileName),
FileMode.Open);
            StreamReader fileReader = new StreamReader(indexFile);
            ArrayList robotFiles = new ArrayList();
            ArrayList myloadedRobots = new ArrayList();
            string lineInput;
            try
            {
                while ((lineInput = fileReader.ReadLine()) != null)
                {
                    robotFiles.Add(lineInput);
                }
            }
```

```
            finally
            {
                fileReader.Close();
                indexFile.Close();
            }

            for (int k = 0; k < robotFiles.Count; k++)
            {
                try
                {
                    fileName = (string)robotFiles[k];
                    indexFile = new FileStream((directory + fileName),
FileMode.Open);
                    fileReader = new StreamReader(indexFile);
                    byte robotID = byte.Parse(fileReader.ReadLine());
                    PhysicalRobot toBeAdded = new PhysicalRobot(robotID);
                    toBeAdded.setName(fileReader.ReadLine());

toBeAdded.setCommAddress(short.Parse(fileReader.ReadLine()));
                    myloadedRobots.Add(toBeAdded);
                }
                catch (FileNotFoundException e)
                {
                    string temp = e.StackTrace;
                    frmRobotFileNotExist error = new frmRobotFileNotExist();
                    error.ShowDialog();
                }
                finally
                {
                    fileReader.Close();
                    indexFile.Close();
                }
            }
            return myloadedRobots;
        }
        private void initRadio()
        {
            try
            {
                serialComm.Open();
                serialComm.ReadTimeout = 100;
                serialComm.WriteTimeout = 100;
            }
            catch
            {
                MessageBox.Show("The Server Radio is not properly
configured.");
            }
        }

        private void updateAnalogs(ref PhysicalRobot myRobot)
        {
            string address = myRobot.RobotCommAddress2().ToString();
            try
            {
                serialComm.Write(address + "00100");
                string[] response = serialComm.ReadLine().Split('|');
```

```
            if (response.Length == 10)
            {
                myRobot.setAnalogs(ref response);
            }
        }
        catch
        {
            tmrUpdate.Stop();
            MessageBox.Show("Lost Communications");
        }
    }
    private void updatePWMS(ref PhysicalRobot myRobot)
    {
        string address = myRobot.RobotCommAddress2().ToString();
        try
        {
            serialComm.Write(address + "00200");
            string[] response = serialComm.ReadLine().Split('|');
            if (response.Length == 6)
            {
                myRobot.setPWMS(ref response);
            }
        }
        catch
        {
            tmrUpdate.Stop();
            MessageBox.Show("Lost Communications");
        }
    }
    private void updatePosition(ref PhysicalRobot myRobot)
    {
        string address = myRobot.RobotCommAddress2().ToString();
        try
        {
            serialComm.Write(address + "00300");
            string[] response = serialComm.ReadLine().Split('|');
            if (response.Length == 5)
            {
                myRobot.setPosition(ref response);
            }
        }
        catch
        {
            tmrUpdate.Stop();
            MessageBox.Show("Lost Communications");
        }
    }
    private void updateData(object o, EventArgs eargs)
    {
        PhysicalRobot currRobot = (PhysicalRobot)loadedRobots[0];
        if (currRobot.getCommStatus().StartsWith("Connected"))
        {
            updateAnalogs(ref currRobot);
            updatePWMS(ref currRobot);
            updatePosition(ref currRobot);
        }
    }
```

```
        //Events
        private void newSimulationToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            frmNewScenario newDialog = new frmNewScenario(ref loadedRobots,
ref serialComm);
            newDialog.ShowDialog();
            envWidth = newDialog.getEnvWidth();
            envLength = newDialog.getEnvLength();
            panelResize();
            btnStartSim.Enabled = true;
            pnlEnvironment.Visible = true;
            newDialog.Dispose();
        }
        private void installedRobotsToolStripMenuItem_Click_1(object sender,
EventArgs e)
        {
            frmConfigureRobots newDialog = new frmConfigureRobots();
            newDialog.setRoboList(ref loadedRobots);
            newDialog.ShowDialog();
        }
        private void robotHealthToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            roboHealth.Visible = true;
            roboHealth.startTimer();
        }
        private void commTestToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            CommTest test = new CommTest(ref serialComm);
            test.Show();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            tmrUpdate.Start();
        }
        private void button2_Click(object sender, EventArgs e)
        {
            tmrUpdate.Stop();
        }

        private void panelResize()
        {
            double tempEnvHeight = envLength;
            double tempEnvWidth = envWidth;
            if (tempEnvWidth >= tempEnvHeight)
            {
                envLength = tempEnvHeight;
                envWidth = tempEnvWidth;
            }
            else
            {
                envLength = tempEnvWidth;
                envWidth = tempEnvHeight;
            }
```

```
                envLength *= 12;
                envWidth *= 12;

            double ratio = envWidth / envLength;
            if (ratio <= (670/400))
            {
                pnlEnvironment.Height = 400;
                pixelsPerUnitD = (400 / envLength);
                pnlEnvironment.Width = (int)(pixelsPerUnitD * envWidth);
            }
            else
            {
                int tempHeight = 400;
                pixelsPerUnitD = (400 / envLength);
                while (pixelsPerUnitD * envWidth > 670)
                {
                    tempHeight--;
                    pixelsPerUnitD = (tempHeight / envLength);
                }
                pnlEnvironment.Width = (int)(pixelsPerUnitD * envWidth);
                pnlEnvironment.Height = (int)(pixelsPerUnitD * envLength);
            }
            pixelsPerUnit = (int)pixelsPerUnitD;
            if (pixelsPerUnit == 0)
                pixelsPerUnit = 1;
        }

        private void btnStartSim_Click(object sender, EventArgs e)
        {
            //While goal not met
            foreach (PhysicalRobot currRobot in loadedRobots)
            {
                int[,] tempMatrix = new int[10, 2];
                tempMatrix[0, 0] = currRobot.getPosX();
                tempMatrix[0, 1] = currRobot.getPosY();
                serialComm.Write(currRobot.getRobotID().ToString() +
"00500");
                string response = serialComm.ReadLine();
                string[] output = response.Split('|');
                currRobot.setIntruderPosition(ref output);
                if (currRobot.getConfidenceInterval()>50)
                {

                }
                //pnlEnvironment.Refresh();
            }
            //After goal is met
            //MessageBox.Show("intruder captured!");
        }

        private void pnlEnvironment_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            GraphicsPath myG = new GraphicsPath();
            Matrix mat = new Matrix();

            foreach (PhysicalRobot currRobot in loadedRobots)
```

```
            {
                if (currRobot.isInSimulation())
                {
                    int x = (int)(currRobot.getPosX() * pixelsPerUnitD);
                    int y = (int)(currRobot.getPosY() * pixelsPerUnitD);

                    if (x >= 0 && y >= 0)
                    {
                        myG = new GraphicsPath();
                        mat = new Matrix();
                        mat.RotateAt((int)(currRobot.getHeading()), new
PointF(x, y));
                        myG.AddRectangle(new Rectangle((x - 2 *
pixelsPerUnit), (y - 4 * pixelsPerUnit), (4 * pixelsPerUnit), (6 *
pixelsPerUnit)));
                        myG.AddRectangle(new Rectangle((x - 3 *
pixelsPerUnit), (y - 5 * pixelsPerUnit), (1 * pixelsPerUnit), (9 *
pixelsPerUnit)));
                        myG.AddRectangle(new Rectangle((x + 2 *
pixelsPerUnit), (y - 5 * pixelsPerUnit), (1 * pixelsPerUnit), (9 *
pixelsPerUnit)));
                        myG.Transform(mat);
                        g.DrawPath(new Pen(Color.Black), myG);

                    }
                }
            }
        }
    }
}
```

Appendix H: Purchased Materials